Robust Lexical Access using Context Sensitive Dynamic Programming and Macro-Substitutions

Peter Nowell

Ph.D.

University of Edinburgh

1991

Abstract

This thesis presents a collection of techniques for automatically extracting and applying phonological knowledge. The phonological knowledge is extracted by statistically analysing the output of a lexical access algorithm. This knowledge is then incorporated into the lexical access algorithm at run time. The following items outline some of the more important and novel features of this approach.

- We have adapted a classical DTW algorithm and chart parser for use in lexical access. This led to a number of improvements such as corrective training, generation of the N-best paths and beam width pruning.

- Phonological knowledge is extracted by statistically analysing the output of the lexical access algorithm. This process is completely automatic and requires no human intervention.

- The phonological knowledge is efficiently incorporated into the lexical access algorithm at run time. The rule application process is completely deterministic since the rules are implemented solely upon the basis of past substitutions.

- The off-line storage is greatly reduced since each template in the lexicon contains a single pronunciation. The additional on-line storage is independent of the lexicon size. These techniques will therefore scale up to large vocabulary speaker independent speech recognition tasks.

- The phonological rule base, lexicon, and phrase structure grammar rules are distinct entities. It is therefore possible to modify one without having to the modify the others.

At the end of the thesis we make some suggestions for extending the work reported here.

Declaration

I declare that this thesis has been written by myself and that the research reported within has been conducted by myself unless indicated otherwise.

Peter Nowell

Acknowledgements

A number of people have made contributions to the research described in this thesis.

I would particularly like to thank the Centre for Speech Technology Research (CSTR) at the University of Edinburgh for providing the speech data which was used during the course of my research, Henry Thompson for providing the source code to MChart which was used to construct the chart parser and Richard Rohwer for providing the source code to his N-gram modelling program known as Treelab. I would also like to thank David McKelvie, my supervisors Henry Thompson and Stephen Isard, and my family and friends for the help, support, and encouragement which they have given.

This project was funded by the Science and Education Research Council

Table of Contents

Chapter 1

Introduction

The construction of machines that are capable of recognising utterances from a wide range of speakers whilst using a large lexicon and a free syntax is an important goal for many speech recognition researchers. In order to achieve this ambitious goal a machine will have to incorporate many different sources of knowledge. These knowledge sources will include the characteristics of speech sounds (phonetics), variability of pronunciation (phonology), the stress and intonation patterns of speech (prosodics), the pronunciations of words (lexicon), the grammatical structure of the language (syntax), the meaning of words and sentences (semantics), and the structure of conversations (pragmatics). Approximations still have to be made in each of these areas because of our limited knowledge of all aspects of speech recognition and understanding

The first stage of most large vocabulary speech recognition systems usually recognises subword units such as phonemes [N. Sugumara & Furui 83], triphones [Lee 88], or syllables [M.J. Hunt & Mermelstein 80]. The type and range of subword units depends upon the design and discriminatory capabilities of the acoustic front end (AFE) and may include around 45 phonemes, 7000 triphones or 20000 syllables. The acoustic front end will typically segment and label a parameterised representation of the acoustic signal to create a lattice of symbols which span the corresponding portions of the acoustic input. The segmentation and labelling techniques are based upon the idealised assumption that it is in fact possible to segment an acoustic speech signal.

In practice it is often impossible to precisely locate segment boundaries because of the effects of coarticulation between adjacent phonemes. This problem is particularly severe when trying to segment shorter constituents such as phonemes since any coarticulation will affect a larger proportion of a phoneme's acoustic characteristics. This problem arises because the pronunciation of one phoneme tends to overlap with that of the next. Even when are significant differences between the acoustic characteristics of the two phonemes such as /p/ and /ii/ it can still be impossible to reliably state where the /p/ ends and the /ii/ begins. The situation is much worse with acoustically similar phonemes such as /i/ and /o/ in 'ratio' where the /i/ phoneme gradually merges into the /o/ phoneme. Even though it is impossible to precisely locate the point at which one phoneme begins and another ends segmentation and labelling techniques are still able to provide useful results.

Even if an acoustic front end (AFE) was able to produce a perfect segmentation and labelling of the acoustic signal there would still be differences between the expected and observed pronunciations of words. Most words can be pronounced in a number of different ways (depending upon the linguistic environment as well as the speaker) and this can lead to even perfect segmentations and labellings of utterances that are quite different.

Of course no AFE is perfect and it is unlikely that one ever will be since even skilled phoneticians cannot always agree on the correct transcription of an utterance. Experiments have shown that phoneticians tend to only agree amongst themselves for about 51% of the time when labelling speech in a foreign language [Shockey & Reddy 73]. Segmentation and labelling errors by an AFE will introduce further transformations of the underlying transcription on top of those introduced by the speaker as shown below in the phrase 'fast speech'.

(1)	/f a s t # s p ii ch/	underlying transcription
(2)	/f a s p ii ch/	pronounced
(3)	/dh i s b i i ch/	recognised

All of these variations makes accurate and robust lexical access extremely difficult since the underlying transcription, i.e. the transcription corresponding to the standard citation form pronunciation of the individual words, will have undergone many transformations before it reaches the lexical access component of a speech recognition system. Fortunately many of the variations appear to be systematic and can be captured by systems of phonological rules such as those described in papers by Cohen and Mercer [Cohen & Mercer 75] and Oshika [Oshika *et al* 75] which can be used in speech recognition systems.

The phonological rules describe the kind of variations that are commonly encountered by a speech recognition system when processing continuous human speech. These will include variations which occur during the production of speech (human phonology) as well as during the recognition of speech (machine phonology). A continuous speech recognition system which is intended to handle a large vocabulary and multiple speakers will have to be able to capture these phonological variations if it is to be able to successfully mediate between the lexical entries and their surface realisations.

Phonological rule systems have been constructed over a period of many decades ([Gimson 70], [Ramsaran 90] etc.) and are still being developed and extended by linguists and phoneticians. The standard symbol vocabulary which is used to express these rules is the International Phonetic Alphabet (IPA) which allows fine phonetic distinctions to be expressed that are able to describe subtle variations in pronunciation. Unfortunately the inventory of current AFEs [1] will almost certainly be different from that which will have been used by the linguists and phoneticians when constructing the rules.

Machine phonology will introduce further modifications on top of those already introduced by human phonology. However machine phonology has not been investigated as extensively as human phonology since it is largely dependent upon the

[1]The inventory of our AFE is the 'Machine Readable Phonemic Alphabet' (MRPA-bet) which is listed in the appendix on page A.1.

implementation details of the AFE. In general rule systems for machine phonology do not exist.

The phoneme inventory of an AFE will almost certainly be different from that which was originally used by the linguists and phoneticians to express the phonological rules. The alphabet of the AFE might contain symbols representing different entities such diphones or triphones or a more limited set of phonemes and allophones. The inventory of the AFE may contain many allophones where it is able to make fine phonetic distinctions and fewer broad class phonemes where such distinctions are not possible. It will therefore be necessary to translate the rule systems into the language of the AFE by modifying and omitting rules in order to match the specificity of the rules with the discriminatory capabilities of the AFE. It would not be sensible for example to incorporate rules accounting for the nasalisation of vowels before nasal consonants if the AFE is unable to distinguish between nasalised and non-nasalised vowels. The selection and translation of the rules is likely to be time consuming and tedious process which will have to be repeated for each instantiation of a particular AFE.

In this thesis we hope to take a small step towards the goal of large vocabulary, speaker independent, speech recognition by focusing on the automatic extraction and run time utilisation of phonological knowledge within the lexical access component of a speech recognition system. The lexical access component mediates between the standard citation form pronunciation(s) of words stored in the lexicon and the (potentially erroneous) phonemic transcription produced by the AFE which segments and labels the parameterised speech signal. Phonological 'rules' are automatically extracted by analysing the output of the AFE using a computer which has been programmed to look for variations in pronunciations arising from human and machine phonology. The rules which are extracted in this way are necessarily compatible with the discriminatory capabilities of the AFE since they are extracted by analysing its output. No time consuming and tedious human intervention is required to either extract the rules, remap them into the vocabulary of the AFE, or incorporate them into the speech recognition process.

1.1 A Historical Perspective

The ARPA (Advanced Research Projects Agency) speech understanding project was set up in the early 1970's with the objective of obtaining a breakthrough in current speech understanding capabilities that could be exploited by practical speech recognition systems [2] [Klatt 77]. The majority of the systems which were developed during this period used explicit phonetic, phonological, syntactic, and semantic knowledge in an attempt to extract the underlying utterance from the acoustic signal. In the HWIM system for example, a number of parameters such as formant frequencies, spectral energy densities and zero crossing rates were calculated. These parameters were then processed by a number of rules (actually BCPL code fragments) to identify and label potential segments which were then refined by the application of further rules. A simple rule taken from the final report describing the AFE [Woods 76] is paraphrased below. This rule was used to refine the labels of previously detected plosives (PLOS).

```
Rule 8d:  vplos
  Segments labeled as PLOS are checked for a burst towards the
  end. If one is found, and the voice onset time (VOT) is more
  than 20mS, then an optional unvoiced plosive is added. If the
  VOT is small and there does not appear to be any frication on
  either side which would shorten the VOT of an unvoiced plosive,
  then the label is changed to a voiced plosive label.
```

Although the performance of the acoustic front ends were particularly poor at that time (HWIM phoneme recognition accuracy was 52%) it was believed

---

[2]The ARPA requirements for a successful system were that it should be able to recognise and understand with less than 10% semantic error, continuous speech from many cooperative speakers using a lexicon of 1000 words and domain dependent syntax.

that the use of explicit constraints provided by higher knowledge sources would overcome many of the limitations. Unfortunately this did not turn out to be the case. HARPY [Lowerre & Reddy 90] was the only system which satisfied the ARPA requirements [Klatt 77] and this was achieved by pre-compiling all of the knowledge sources into a single large network which spanned all legal utterances. Recognition was then achieved by finding the path through the network which best matched the output of the acoustic front end.

The inability of knowledge based approaches to reliably segment and label the acoustic speech signal led to the abandonment of these approaches in favour of mathematically well-defined pattern matching techniques such as dynamic time warping [Vintsyuk 68], [Sakoe & Chiba 78], hidden Markov models [Baker 75], and more recently neural networks [Waibel *et al* 89]. The performance of these techniques have been sufficiently high that many researchers have found them to be an attractive alternative to explicit knowledge based approaches.

The first systems to incorporate pattern matching techniques used whole word templates containing vectors of speech parameters. A training process was used to construct the templates which were then time aligned against a similarly parameterised speech signal during recognition. Word templates were found to yield the best recognition rates since they naturally encapsulated within word phonological variations. Different pronunciations could also be handled straightforwardly by using multiple acoustic templates for each word.

However, as the size of the vocabulary and the diversity of the speakers increase whole word modeling becomes increasing impractical. Training data cannot be shared amongst acoustically similar templates so each word model has to be trained individually. This places great demands upon the patience of the user who must provide the training utterances. Despite their near term success researchers have come to the conclusion that these methods will not scale up to more complex tasks.

Many recent speech recognition systems have attempted to overcome these problems by modelling sub-word units such as syllables, triphones, diphones, or phonemes in place of whole words. Although each sub-word model still contains

vectors of speech parameters the word templates now contain a network of symbolic labels which index the sub-word models. Different paths through the network represent different pronunciations of words. The networks are pre-compiled by the application of phonological rules which add additional branches for those variations which are expected to commonly occur in continuous fluent speech. Significant savings in storage and computational requirements are obtained and the training requirements are substantially reduced due to the sharing of sub-word models.

Although this represents a significant improvement we believe that it only represents a short term solution to the problem. A rigorous application of phonological rules has been found to result in a large number of pronunciations being generated. In one experiment for example [Baker 75], over 430 different pronunciations of a single word 'always' were derived. It is obviously impractical to store and process the networks which such a rigorous application would produce. In practice speech recognition systems can consider only the most common phonological variations which occur during speech production. Even so, the use of pre-compilation techniques still results in large and complex networks, especially when word boundary phenomena are taken into account.

1.2 Variability in Speech Production

It soon becomes apparent to anybody who studies human speech that a great deal of variability is present, even when identical phrases are spoken by the same person. There are variations between the pronunciation of words and phrases when uttered by speakers from different geographical backgrounds (accentual variations), variations which are dependent upon the social context (stylistic variations) and variations which exist between words spoken in isolation and within continuous speech (fast speech variations). These variations may be speaker independent, such as fast speech phenomena which occur in everybody's speech, or speaker dependent and exist only in the speech of a single speaker or group of speakers such as accentual variations. In addition to the variations introduced during the

production of speech, an acoustic front end will also introduce variations (errors) during segmentation and labelling. Likewise, some of these errors are more likely to be introduced by all AFEs (machine independent) whilst others will be more likely to be associated with a particular front end (machine dependent).

All of these variations, whether they be introduced during production (human phonology) or recognition (machine phonology) cause problems for lexical access algorithms since they obscure the mapping between pronunciations stored in the lexicon and the transcriptions produced by the AFE. In the following sections we will briefly describe some of the variations and the implications that they have for lexical access. We will then describe the format of phonological rules which describe the variations and show how these relate to the macro-substitutions which we will be using in our lexical access algorithm.

1.2.1 Human Phonology

Accentual Variations

Spoken English can be classified into a number of varieties (or accents) according to the geographical background of the speakers. Different accents have distinctive variations in pronunciations. Non-native speakers often sound strange or unnatural when they use the phonemes and phonological rules of a foreign language to pronounce English words. There are also obvious differences in the speech of native speakers from different countries (for example American-English and British-English) and also amongst people from different regions within a country (e.g. Liverpool and Birmingham). Some of the variations can be so extreme (i.e. broad accent) that it can be difficult for speakers from different regions to understand one another.

The standard accent of British-English is known as Received Pronunciation (abbreviated 'RP'). This is not the accent of any particular region but the accent of those in the upper reaches of the social scale. It is also the most widely studied of all British accents. However, the majority of speakers will have some kind of regional accent. Regional accents differ from RP both in their phoneme inventory

and the phonological processes which tend to occur. For example most English accents allow a pre-vocalic /r/ in the pronunciation of isolated words although they vary in whether they allow the pronunciation of /r/ after vowels. RP does not have a post-vocalic /r/ although Scottish and Irish accents (and most North American accents) do. However /r/s can still occur after vowels in RP since it is possible for /r/ phonemes to be inserted at word boundaries if the following word begins with a vowel. This example shows that many of the phonological rules that are used during the production of speech are dependent upon the accent of the speaker.

Stylistic Variations

Variations in the pronunciation which occur in response to the situations in which speakers find themselves are termed stylistic. The most important factor in conditioning these variations is the degree of (in)formality as perceived by the speaker. The degree of formality depends upon many factors which include the relative status of the person who is being spokɛn to, how well they know each other, what they are talking about, and the actual location of the conversation. In a formal situation a speaker will tend to articulate sounds with more care, individual sounds are more likely be precisely articulated and relatively fewer phonemes will be reduced or omitted. However, in informal situations the speech will be quicker and articulated with less effort and many more phonemes will be reduced and omitted. A casual style of pronunciation should not be considered to be less correct than a careful, precise pronunciation, it is really just a matter of appropriateness.

All speakers will produce utterances that exhibit stylistic variations although the degree of variation will vary amongst contexts and speakers. Speech input to a computer tends to occur in a fixed social context so the stylistic variations which do occur are likely to be relatively constant, at least for a particular speaker. Some people will speak naturally (casually) to machines whilst others may pay particular attention as to how they pronounce words. As people become used to conversing with the machine their speech will tend to become more relaxed or 'lazy'.

Fast Speech Variations

The pronunciation of words often differs when they are spoken in isolation and in continuous speech. In continuous speech words are spoken quicker and with greater fluidity than in isolation. As the description 'continuous speech' implies, there are also no pauses between words or any other easily observed and reliable indicator of word boundaries. Furthermore, many phonemes which occur at the boundaries tend to merge together whilst others may be omitted. For example, there is typically only one /s/ phoneme and no /t/ phoneme in a continuous pronunciation of the phrase 'fast speech'.

Taking 'African' as an example the following progressive modifications are possible with an increasing rate of articulation [Harrington *et al* 86].

African	
/a f r i k @ n/	Slow, careful pronunciation
/a f r i k n/	
/a f r @ k n/	
/a f r k n/	
/a f r k ng/	Fast production

The merging and deletion of phonemes at natural (i.e. fast) speech rates are thought to be caused by the inertia of some speech actuators which cannot keep up with the desired phoneme rate. Often there will not be sufficient time to articulate one phoneme precisely before the pronunciation of the next phoneme must begin. As many of these speech reductions are the result of the physical limitations of the speech production apparatus they are also likely to be speaker independent. The frequency and the nature of the reductions will be dependent to some extent upon the speaker as well as the speech rate.

1.2.2 Machine Phonology

Machine phonology is concerned with the phonological phenomena that are introduced during the acoustic phonetic recognition process. These phonological

variations, as well as straightforward recognition errors, occur in addition to, and on top of those variations which occurred during the speech production process as a result of human phonology. Like human phonology, machine phonology can also be classified according to whether the variations are machine independent or machine dependent.

Machine independent variations will be introduced when the acoustic realisations of phonemes have a greater similarity in some contexts than in others. For example, the burst frequency of a /k/ is usually lower than that of a /t/ although the burst frequency of a /k/ before a front vowel is similar to the burst frequency of a /t/ before a back vowel. The similarities in the acoustic realisations ensure that acoustic front ends will be more likely to confuse a /k/ before a front vowel with a /t/ before a back vowel and vice versa. Machine dependent variations will also be introduced by implementation peculiarities and limitations of the front end.

1.2.3 Implications for Lexical Access Algorithms

The effect of phonological variations is to introduce a complex and many to one mapping between the transcription of an utterance produced by an acoustic front end and the idealised sequence(s) of phonemes produced by concatenating the corresponding word templates. It is generally believed [Klatt 77] that the end result of these phonological phenomena is to introduce ambiguity so that even the best lexical access routines will propose many words that were not in the original utterance but which occurred as a result of fortuitous matches.

The following example is taken from Klatts review of the ARPA speech understanding project and highlights the difficulties which lexical access procedures face when dealing with continuous speech. The American pronunciation of the phrase 'Did you hit it to Tom?' has been transcribed using the MRPAbet phonemes. The first transcription shows the idealised sequence of phonemes which is produced by concatenating the individual word templates. The sequence of phonemes in the

second transcription shows how the phrase might actually be pronounced in continuous speech.

/# d i d # y uu # h i t # i t # t uu # t o m #/

/d i j @ h i i t @ t o m/

This example was intended to show that lexical access will be difficult because, even if the AFE was perfect there are would still be extensive variations in the way that words are pronounced in continuous speech. In this example the modifications are the result of the following allophonic processes. Unfortunately the set of phonemes in the MRPAbet provides no way of representing the flap which occurs as a consequence of rule (3).

1) Palatalisation of /d/ before /y/ in 'did you'
2) Reduction of unstressed /u/ to schwa in 'you' and 'to'
3) Flapping of intervocalic /t/ in 'hit it'
4) Assimilation of /t/ in 'it to'

It was claimed that phonological phenomena such as these cause valuable distinctions to be lost[3] since it is no longer possible to distinguish a palatalised /t/ from a /j/, an unstressed /u/ from a /@/ and so on. Modifications such as these make it more difficult to hypothesise lexical candidates given an input transcription since each phonological rule results in irreversible ambiguity. The phonological transformations do not in general have unique inverses that could be used to uncover the underlying sequence of phonemes. A surface /@/ for example could be

[3]This point is somewhat controversial since others [Church 87] would argue that phonological processes provide additional phonetic cues which can be used to aid recognition - see section on Phonological Parsing

produced by the reduction of almost any sufficiently destressed vowel. The situation will be even more complicated with real AFEs since many of the phoneme recognitions will be uncertain or incorrect. In this case it is also possible that the /@/ resulted from the incorrect recognition of absolutely any phoneme.

1.3 Phonological Rules and Phonotactic Constraints

Much of the phonological variation appears to be consistent and is governed by phonological rules which are conditioned by the presence of surrounding vowels, consonants and stress patterns. Speech researchers, linguists, and phoneticians have developed systems of phonological rules for capturing the appropriate generalisations. A typical phonological rewrite rule expressed as follows

```
t -> ! / V_V
```

predicts that a /t/ will be deleted when it occurs in an inter-vocalic context. Systems of rules such as these have been proposed to explain accentual variations, stylistic variations and fast speech phenomena as well as machine dependent and independent variations. These rules can be classified into three groups according to whether they result in the insertion, substitution or deletion of phonemes.

Insertion rules allow a single segment to be inserted and are comparatively rare in fast speech. One example is the linking /r/ rule which would insert a /r/ between 'require' /r i k w ai @/ and 'any' /e n ii/ in continuous speech to produce /r i k w ai @ r e n ii/.

Alternation or Substitution rules allow phonemes in the citation form to be substituted by other different phonemes. For example, the substitution of the /ʊʊ/in /t uu/ by a '@' would be produced by an alternation rule.

Deletion rules allow for a single segment to be deleted. In the case of vowels, only the /@/ can be deleted, thus giving /a f r i k n/ as a possible pronunciation of /a f r i k @ n/.

The application of these rules is dependent upon the surrounding context. The inclusion of word boundaries (#), syllable boundaries (.) and stress markers would seem to be necessary to ensure their correct application. For example, the alternation rule which substitutes /i/ in unstressed syllables with /@/ can apply in all environments except word final [Harrington *et al* 86].

Therefore we can have

	/# p r o . b l i m #/	problem
reducing to	/# p r o . b l @ m #/	

but not

	/# s i . t i #/	city
reducing to	/# s i . t @ #/	word final /i/
or	/# s @ . t i #/	stressed syllable

There are also restrictions in English on the permissible sequences of phonemes which are governed by phonotactic constraints [Harrington *et al* 87]. For example, if a word begins with three consonants then the first phoneme must be a /s/ followed by a stop /p, t, k/ and an approximant /l, r, y, w/. Knowledge of these constraints could be used to provide clues about the location of word boundaries. The phoneme sequence /m g/ in /m g l/ for example does not satisfy the phonotactic constraints of English. A word boundary must therefore be located after the first phoneme thus giving the sequence /m # g l/ as in 'same glass'. It is only possible to use such constraints if they are not violated by reduced speech forms. However some sequences which would be impermissible in the citation form pronunciations do occur in continuous speech. The use of such constraints would be further hampered by mis-recognitions made by the AFE since these may cause many phoneme sequences to be ruled out on the basis of incorrect labellings.

1.4 Phonological Rules as Macro Substitutions

Below we show a small example of common phonological phenomena which includes examples of human phonology and machine phonology. The phonological rules describe mappings from the standard citation form pronunciation to pronunciations within fluent connected speech and show how phonemes may be deleted, substituted, or inserted, in continuous speech. The realisation of a particular phoneme may be dependent upon the preceding left context, following right context, or a combination of left and right context which we call centre context.

These mappings can also be equivalently described by what we call macro-substitutions (MSubs). These are sequences of individual phoneme substitutions (X.Y) that show how phonemes (X) in the citation form pronunciation are realised as (i.e. substituted with) phonemes (Y) in the recognised surface form pronunciation. The 'null phoneme' /-/ has zero duration so substitutions of it with citation form and surface phonemes correspond to deletions of the respective phonemes. In the example below, the palatalisation rule d ↦ j /_ # y states that a /d/ can be realised as a /j/ when it is followed by a word boundary /#/ and /y/. The corresponding set of substitutions also shows how the /d/ and /j/ phonemes are substituted with one another in the appropriate context.

Human Phonology	Macro-Substitution	
j u ↦ u / # n _	(#.-)(n.n)(j.-)(u.u)	; Deletion
d ↦ j / _ # y	(d.j)(#.-)(y.y)	; Palatalisation
! ↦ t / n _ s	(n.n)(-.t)(s.s)	; Stop Insertion
Machine Phonology		
p ↦ g / _	(p.g)	; General recognition error
i@ ↦ i @ / _	(i@.i)(-.@)	; Mis-segmentation of diphthongs
l ↦ l l l/ _	(l.l)(-.l)(-.l)	; Mis-segmentation of liquids

The phonological rules predict that some particular phonemes will be realised differently in certain contexts. We should therefore expect that the probability of some phoneme substitutions (such as (d.j)) will be significantly higher when they occur in contexts where phonological rules are applied. When this situation arises the sequences of individual phoneme substitutions will be positively correlated and should therefore be treated as a whole i.e. as a macro-substitution rather than a sequence of independent substitutions. The degree of correlation will be related to the extent to which the rule can be considered optional, compulsory rules being more highly correlated than optional rules. As we shall see it is possible to automatically determine the sequences of phoneme substitutions that are significantly correlated which is in many ways equivalent to determining the phonological processes that are occurring in the speech being analysed.

1.5 System Outline

The speech recognition process takes place over three distinct levels, these being phoneme recognition, word recognition and sentence recognition. Figure 1–1 shows the major components of the current lexical access system. Boldly outlined boxes contain the individual modules, whilst rounded rectangles indicate various sources of data. The density of the connecting arrows indicates the degree of coupling between the various modules. Dashed arrows indicate loose coupling where the components communicate via messages and may be executing on physically distinct processors. Dotted arrows indicate off-line processing of data.

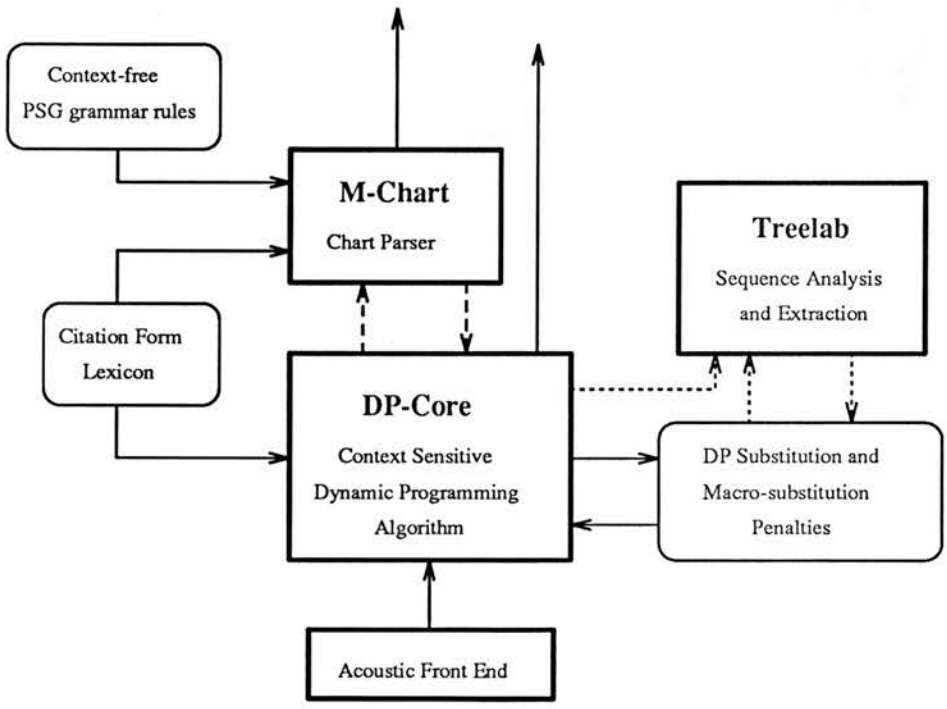


Figure 1–1: System Architecture

The acoustic front end produces a lattice of phoneme segments. Each segment contains a number of different phoneme candidates (typically 6) which are ranked according to a likelihood measure of their presence. The lattices are well structured in the sense that there are no overlaps or gaps between adjacent segments. The

phoneme lattices, as well as lexicons and transcriptions, were provided by the Centre for Speech Technology Research (CSTR), University of Edinburgh.

Word level recognition is achieved by using a dynamic programming algorithm which determines the optimal alignment(s) between the phonemes in a lattice and sequences of word templates. The lexicon consists of a number of word templates each of which contain a single phonemic transcription of the the citation form pronunciation of the word. Simple variations between the citation form and recognised surface forms are handled robustly by the dynamic programming algorithm. Context sensitive 'macro-substitutions' can be used to handle phonological processes which occur within continuous speech (human phonology) as well as systematic defects in the acoustic front end (machine phonology).

The DP alignments from previous (mis)-recognitions are processed off-line by Treelab which statistically analyses the alignments and uses N-grams to model the sequence probabilities [Rohwer 87]. A tree containing all previously observed phoneme substitution sequences is constructed and then a minimal sub-tree containing sequences of significantly correlated phoneme substitutions is automatically extracted from it. The effects of phonological rules will be realised as sequences of correlated phoneme substitutions. The probability of these context dependent substitutions can not be accurately calculated by the basic DP algorithm. We use the correlated sequences at run-time in our context-sensitive DP algorithm to calculate corrected alignment scores which overcome the effects of phonological processes.

An optional chart parsing module can be used to constrain the ordering of templates during the DP alignment process. The parser ensures that only syntactically valid parses are extracted from the phoneme lattices. The chart parsing module is an extension of MChart [Thompson 83] which uses a dynamic programming algorithm to find the optimal parse(s) in a word lattice. The parser operates time synchronously from left to right and in parallel with the DP alignment algorithm, synchronisation and communication is achieved by passing messages. The parser's syntactic knowledge is in the form of context-free phrase structure rules. These rules are implemented directly without the usual conversion to a comparable finite

state network and can be modified without having to make changes to either the chart parser or the DP alignment module.

The outline of the thesis is as follows. In the next chapter we briefly discuss other approaches that have been used to incorporate phonological knowledge into lexical access algorithms. Then, starting from Chapter 3 we will describe various aspects of our approach. Chapter 3 introduces the concept of dynamic programming and the modifications which had to be made to adapt a traditional dynamic time warping algorithm for use in lexical access. In Chapter 4 we describe the enhancements and extensions that have been made to the basic DP alignment algorithm. These include corrective training, efficient generation of the N-best paths and an investigation into the potential benefits of beam-width pruning. Chapter 5 introduces the chart parser and explains how it communicates and synchronises its activity with the dynamic programming module so as to constrain the alignments to those which are syntactically valid. The theory which lies behind the use of macro-substitutions to account for context sensitive phoneme substitutions is explained in Chapter 6. In Chapter 7 we describe how the macro-substitutions can be efficiently implemented at run time within the existing dynamic programming algorithm. Finally in chapter 8 we present the results of our research and describe the relative advantages and disadvantages of our approach. This chapter also contains the conclusions and somes ideas for further research.

Chapter 2

Review

In this chapter we will outline, in roughly chronological order, four different methods which have been used to incorporate phonological knowledge into lexical access algorithms with varying degrees of success. The first section describes an attempt to perform the phonological transformations in reverse by adding additional segments to the phoneme lattice prior to lexical access. This approach encountered a number of serious problems and has been abandoned in favour of pre-compilation techniques which use the rules in their natural generative direction to expand the lexicon with alternative pronunciations. Lexical precompilation is still the most successful technique but it is unlikely to be practical for speech recognition systems which have large lexicons and recognise speech from numerous speakers with various accents. The two most recent techniques which are described involve systems which integrate the phonological knowledge directly into the lexical access algorithms at run time using either context free grammar rules or finite state transducers.

2.1 Lattice Expansion

During the first two years of the ARPA speech understanding project the designers of the BBN HWIM speech understanding system developed two different mechanisms for applying phonological rules [Woods 76]. The first was to analyse the

contents of the phoneme lattice and apply rules in reverse to undo phonological transformations by inserting branches into the lattice (lattice expansion) whereas second mechanism was to apply rules in their normal generative direction to entries in the lexicon and thereby generate alternative pronunciations (dictionary expansion). It was envisaged that some rules would be best implemented in the reverse direction whilst others would be best implemented in the forward direction. However, a number of problems arose when they tried to apply phonological rules in the reverse direction. These problems are summarised below with reference to Figure 2–1, taken from Klatts 'Review of the ARPA Speech Understanding Project' [Klatt 77]. This figure shows the kind of segment lattices produced by HWIM's acoustic front end.

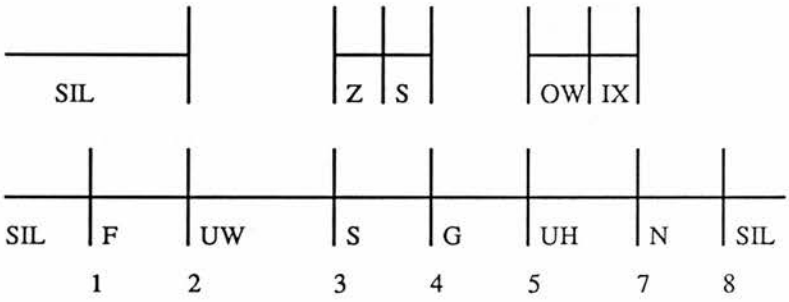


Figure 2–1: A BBN segment lattice for the utterance 'Who's going?' Alternative segmentations are displaced vertically, while each label represents the top choice among a set of 71 phonetic categories. Similarity scores are computed for all possible labels at each segmentation point. For example, the segment starting at boundary (1) is labelled "F" but the correct phonetic segment "H" was the fourth best label choice.

The phonological rules that were used in the HWIM system were collected during the course of their research. The rules reflected phonological phenomena that were seen in the acoustic data as well as common phenomena that would be expected to occur. The rules accounted for both general phonological effects (human phonology) as well as effects that were dependent upon the performance of the acoustic front end (machine phonology). Optional rules were used to account for the specific AFE shortcomings whilst obligatory rules were used for general

phonological effects. Obligatory rules are rules which can be assumed to be always applicable in natural continuous speech. An example of such a rule would be /s t # s/ $\mapsto$ /s/ which rewrites the string of phonemes /s t # s/ with the single phoneme /s/.

They discovered that there was no good way to implement obligatory rewrite in the reverse direction since it was not possible to remove paths matched by such rules without killing off other paths that were not matched. For example, an obligatory rewrite rule such as /s t # s/ $\mapsto$ /s/ can not treated as an obligatory rule in the reverse direction. Many of the /s/ phonemes in a lattice (e.g between segments (3) and (4) in fig. 2–1) will not have arisen from mechanisms corresponding to the rewrite rule and therefore should not be rewritten as /s t # s/. Furthermore, rewriting /s/ phonemes as /s t # s/ will prevent other rules from being used which may have also been applicable.

These problems led them to conclude that all obligatory rewrite rules needed to be treated as though they were optional when applied in the reverse direction. The application of optional rules caused additional branches to be added to the lattice rather than the replacement of existing branches. The net result of treating all rules as being optional was to increase the size and complexity of the lattice as well as the uncertainty about its contents. This increased the amount of subsequent processing that was required to align the word templates.

Further difficulties were introduced due to the fact that the identification of phonemes by the AFE was tentative and often incorrect. Many of the rule applications occurred as a result of fortuitous matches between the lattice and the phonemes on the right hand side of the rules. As a result the rules were applied in all sorts of circumstances that they would not have faced in the forward generative direction. There was also no obvious way to assign scores and durations to the segments that were inserted into the lattice when a rule was successfully matched. The scores of the new segments should be dependent upon the sequence of phonemes which caused the rule to be applied whilst also being consistent with the scores produced by the acoustic front end for the other phonemes in the lattice.

These problems, along with the discovery of efficient solution to the major

problem of generative rules (described in the next section), eventually led them to abandon lattice expansion and use only forward generative rules in the final system. This effectively overcame all of the problems previously described. Obligatory rules could be applied without problem in their intended generative direction and since phonemes were no longer added to the lattice, scores and durations did not have to be calculated for them.

2.2 Pre-Compilation

One of the most successful ways of incorporating phonological knowledge into a speech recognition system is to use the rules in the forward generative direction to pre-compile the base form pronunciations in the lexicon. Within word phonological effects are easily handled by expanding the dictionary off-line to generate a network of possible pronunciations for each word in the lexicon as shown in figure 2–2. This representation has the advantages of providing a compact means of representing alternative pronunciations which can be efficiently matched against the phoneme lattices at run time.

Each of the templates in the lexicon initially contains a single base-form labelling which defines the standard pronunciation of the word. Phonological knowledge in the form of generative rewrite rules is then applied off-line to expand the network by adding alternative paths which cater for possible variations in the pronunciation. Unlike the phonemes which occur in the lattice, those which make up the pronunciations of each word are neither tentative nor potentially incorrect so the problem of fortuitous matches does not arise. Obligatory rewrite rules also no longer present a problem since the combination of the phonemes in each word template along with those in the left hand side of each rewrite rule are always sufficient to ensure that the rules are only applied under the correct circumstances. However, there is a problem which arises when one attempts to apply phonological rules which cross word boundaries. Woods et al. [Woods 76] describe the problem as it occurred in the HWIM system as follows.

It first appeared that the across-word phonological rules could not be applied to a word until one knew its context. Since the set of all possible sentences that could be constructed is potentially infinite, the complete string of phonemes to which one would like to apply rules does not exist until a complete sentence has been hypothesised. Yet, one may need to have applied rules to a word in order to know whether it has occurred in the first place. The problem appeared to be a vicious circle. However the circle can be broken by a technique developed by Klovstad [Klovstad & Mondshein 75] which permits across-word-boundary phonological rules to be applied when the dictionary is being constructed. Within word matches (including those with word boundary effects) can be found at runtime and the contextual constraints implied by the use of a rule are automatically propagated to adjacent words.

This CASPERS linguistic analysis system [Klovstad & Mondshein 75] overcame this problem by compiling the word boundary spanning phonological rules into the lexicon. The structure of the lexicon and the routines which access it ensure that the contextual constraints which arise from the application of a word boundary spanning rule are automatically propagated to adjacent words. This has the desired effect of automatically applying phonological word boundary rules during successive passes through the lexicon. For example, the simple lexicon and phonological rules shown below would be realised as the network in fig. 2–2.

Template	Phonemes	Template	Phonemes	Rules
had	/h a d/	last	/l a s t/	/d # y/ $\mapsto$ /j/
hand	/h a n d/	label	/l ei b @ l/	/n d # l/ $\mapsto$ /n # l/
you	/y uu/	list	/l i s t/	
		likely	/l ai k l ii/	

A terminal node is associated with the last phoneme of each word template which identifies an entry point for succeeding templates. The entry points are determined by the word boundary rules and point back into the lexicon as specified

by the right phonemic context. For example, consider the entry point labelled 2⇒, there are 3 options.

1. The path /n j/ corresponds to the application of the rule /d # y/ ⇒ /j/. This gives an alternative pronunciation of 'hand you' /h a n d y uu/ as /h a n j uu/.

2. The path /n l/ corresponds to application of the rule /n d # l/ ⇒ /n l/. This gives an alternative pronunciation of 'hand label' /h a n d # l ei b @ l/ as /h a n l ei b @ l/.

3. The path /n d/ allows for the standard or citation form pronunciation of 'hand label' as /h a n d l ei b @ l/.

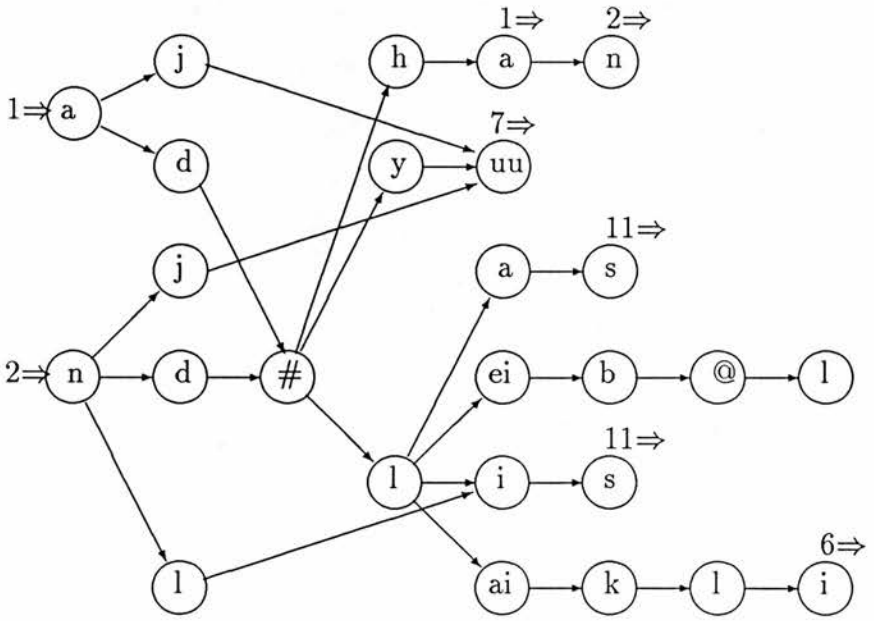


Figure 2–2: A Simple Lexical Network

Further knowledge can be incorporated by assigning probabilities to both the within and between word transitions and also to the phonemes that are matched

with the incoming speech. The resulting network is in effect equivalent to a Markov model (HMM) where the probability of any path through the network represents the likelihood of that particular pronunciation. The probabilities can be estimated by training on a suitable set of data. A finite state grammar could also be incorporated into this network by constraining the set of entry points. The entry points would then only point to words that can follow the current word as defined in the grammar [Jelinek 81]. This complicates the network but reduces the number of word candidates that have to be considered at each stage. This increases the probability of finding the correct path through the network at the expense of a more restricted dialogue.

Expanding the lexicon in this way incurs a clear cost in storage requirements. In the HWIM system for example, the application of phonological rules to the dictionary of 1138 roots and irregularly inflected forms generated a total of 8642 entries in the expanded lexicon even though many of the more exotic rules such as vowel raising were left out. The storage requirements could be expected to increase substantially if the dictionary was expanded to incorporate a more complete set of phonological rules. In order to keep the size of the lexicon manageable it was necessary to restrict the scope of the rules to those which represent the most commonly observed phenomena.

These techniques do not scale up satisfactorily to large vocabulary speaker independent speech recognition since the networks rapidly become excessively large and complex. In one system [Harrington *et al* 86] the application of 450 simple phonological rules to a lexicon of 4000 citation form words produced an additional 5300 reduced form word templates whilst a more rigorous application of rules has derived over 430 possible pronunciations for a single word such as 'always' [Baker 75]. A similar and further increase in the complexity of the lexicon can also be expected when word boundary rules and finite state grammars are incorporated.

The inability of pre-compilation approaches to scale up to substantially more complicated tasks has led some researchers to consider alternative approaches which apply the phonological rules at run time. Two alternative approaches are now described in the following sections.

2.3 Phonological Parsing

In his thesis 'Phonological Parsing in Speech Recognition' [Church 87] Kenneth Church proposes an alternative model of phonological processes which distinguishes between variant features (such as aspiration and flapping) which are dependent upon the context and invariant features (such as voicing, manner and place of articulation) which are relatively independent of context. The lexical access process is divided into two components, the first component uses a syntax of variant features to parse the utterance at the segmental level, whilst the second component matches the parsed constituents against templates stored in the lexicon. Since the phonological processes are accounted for explicitly during lexical access the lexicon is free of allophonic variations and contains a single transcription for each template.

The phonological processes which give rise to the variant features were expressed in terms of phrase structure rules. A chart parser was then used along with these rules to parse a lattice of phonetic segments into syllables and other larger constituents. The following grammar was given as an example to show how aspiration could be restricted to syllable initial phonemes. Terminal categories such as (un)aspirated_p represent phonemes that would be segmented and labelled by an acoustic front end or human transcriber.

```
utterance -> syllable+
syllable  -> onset rhyme
rhyme     -> peak coda
onset     -> aspirated_p | aspirated_t | ......
coda      -> unaspirated_p | unaspirated_t | .....
```

The phonological knowledge was reformulated so as to be free of transformations such as rewrite rules which would add, delete and modify parts of the transcription. The reformulation transforms the problem of determining which rules to apply and in what order to a context free parsing problem for which a number

of well-defined and efficient solutions exist. As an example the thesis illustrates how, given a detailed phonetic transcription, the sentence 'Did you hit it to Tom' could be broken up into the following constituents (using phoneme symbols from our MRPAbet).

/d i j @ # h i # i # t @ # t o m/

This was achieved by employing constraints about syllable structure such as the fact that /h/ phonemes and aspirated /t/s occur only in syllable initial positions whilst glottal stops occur in syllable final positions. Obviously, lexical access will be much easier once the transcription has been parsed into a number of larger constituents. This approach was used with some success to parse and match hand generated transcriptions produced by a skilled linguist. The system was able to cope with a limited degree of uncertainty and could handle some hand labelled transcriptions produced by a novice spectrogram reader. These transcriptions contained a small number of choices in each segment of the lattice. The parser was however overwhelmed by the uncertainty and mis-recognitions that were present in machine generated lattices and as a consequence the recognition performance was drastically reduced.

The machine generated lattices were taken from the BBN acoustic front end developed during the ARPA speech understanding project. These lattices had a much higher branching ratio and contained many more confusions than any of the hand generated transcriptions. The program had difficulty in parsing the lattices since the recogniser did not make sufficiently precise or reliable phonetic distinctions to allow many of the rules to be used.

Many of the rules were also applied in a number of undesirable situations due to mis-recognitions which led to fortuitous matches. It was suggested that these problems might be overcome by relaxing the constraints provided by the phonological rules. This would involve matching the specificity of the rules to the discriminatory capabilities of the acoustic front end. It was also proposed to insert mis-recognised or missing segments into the lattice prior to parsing. For example,

if the following phonemes were observed to be commonly missed by the acoustic front end.

/d/ after /n/
/t/ after /f/
/t/ after /s/

It was suggested that an optional /t/ could be inserted after all occurrences of /f/ and /s/. However if this approach were to be implemented it seems likely that similar problems would be encountered to those observed in the HWIM system which also initially tried to insert missing phonemes into the lattice.

Church's thesis argues that phonological processes, rather than being a source of noise provide an important source of information. Allophonic processes were found to provide reliable cues for determining the location of syllable boundaries in hand transcribed speech. Once the transcription had been parsed into syllables it was much easier to match the lattice contents with the entries in the lexicon to recognise the utterance.

Unfortunately, the performance of current acoustic front ends still leaves much to be desired and this seems likely to persist for the foreseeable future. The conclusion of the thesis was that emphasis ought to be placed upon improving the discriminatory capabilities and accuracy of the acoustic front end after which it should be possible to pursue the implementation of higher level constraints.

2.4 Finite State Transducers

The lexical access component of the CSTR-ISTD (Integrated Speech Technology Demonstrator) speech recogniser [McKelvie 91] uses dynamic programming techniques to provide robust performance in spite of inevitable mis-recognitions. Finite state transducers are used at run-time to account for a small number of basic phonological processes [Thompson *et al* 89]. The use of phonological knowledge at run time allows the lexicon to remain relatively compact with only a few

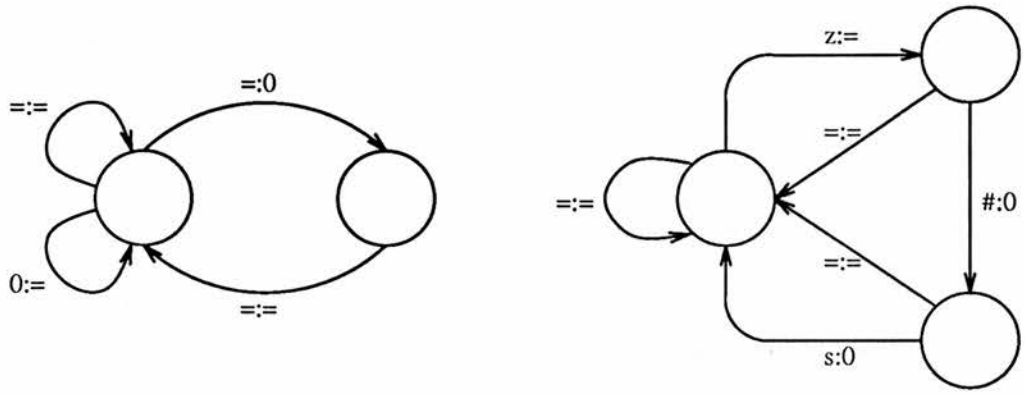
pronunciations per word (usually just one was provided). Phonological rules in the form of finite state transducers are used at run time to provide the DP algorithm with alternative means of generating alignments.

Word recognition is achieved by finding the lowest scoring (most probable) match between sequences of phonemes in the lexical templates and phonemes in a lattice produced by the acoustic front end. A chart parser is used to implement a dynamic programming algorithm with beam-search that obtains a (locally) optimal alignment and word-level segmentation by substituting, inserting, and deleting template and lattice phonemes wherever necessary [Thompson 89]. The individual substitutions which make up the alignment have scores which are calculated from the probability of the input phoneme given the acoustic and the probability of the substitution, insertion or deletion. The dynamic programming algorithm obtains the optimal alignment by finding the sequence of substitutions, insertions, and deletions which gives the lowest overall score.

Non-identity substitutions (i.e. where lexical and surface phonemes differ) will have relatively low probabilities and are penalised according to their relative confusabilities. These probabilities, along with the probabilities of insertions and deletions, are estimated during an iterative training procedure. Insertions are used to add missing phonemes back into the lattice which can then be substituted with the corresponding lexical phonemes. As was the case with HWIM there is the problem of assigning realistic scores and durations to inserted phonemes. This problem was overcome by simply using a constant, pre-determined score and duration which appeared to give satisfactory results. Extraneous phonemes are effectively deleted by substituting the lexical phoneme with a special 'delete phoneme' introduced by the AFE. The 'delete phoneme' is assigned a score which is proportional to the probability that there are no phonemes (i.e. only silence) in the respective segment.

Restrictions on the type of paths which may be used in the alignments are enforced by the use of finite state transducers (e.g. figure 2–3) where labels on the arcs describe the sequence of permissible substitutions that may be used to traverse from one state to another. Each label consists of a pair of symbols separated by ':'.

The first symbol represents a phoneme within a lexical template and the second represents the corresponding surface phoneme. The symbol '=' can represent any phoneme, whilst '#' indicates a word boundary and '0' is the delete phoneme. Further expressive power is possible with the use of prolog type variables (not shown) which, when instantiated, must have the same value on all paths within the transducer.



(a) Context Freee F.S.M. (b) Context Sensitive F.S.M.

Figure 2–3: An example of two simple finite state machines. The first context free machine implements the basic substitutions, insertions, and deletions of phonemes. The second machine incorporates the phonological rule $s \mapsto !/z\#\_$

The finite transducer specifies the syntax of permissible alignments and the dynamic programming algorithm is constrained to those paths that can be 'parsed' by the finite-state transducer and chart parser. The sequences of substitutions, insertions, and deletions which can be used by the basic dynamic programming algorithm are described by the transducer in figure 2–3a. This transducer allows arbitrarily long sequences of substitutions (=:=) and deletions (0:=) but consecutive insertions (=:0) are not allowed. A limited number of phonological processes can be handled relatively straightforwardly by adding additional arcs and states. The simple transducer in figure 2–3b implements a de-gemination rule which allows a higher probability to be used for the deletion of an /s/ phoneme after a word final /z/ phoneme. This phonological process would be implemented by following the arcs z:=, #:0, s:0 during the alignment process.

The system has been used with transducers that represent a relatively small

number of phonological phenomena such as assimilation, gemination and mis-segmentation of phonemes [Williams & Thompson 89] but so far the performance has been unimpressive. The use of context sensitive finite state transducers results in a 2 – 3 times increase in computational requirements with little or no improvement in recognition accuracy [McKelvie 90].

Chapter 3

Dynamic Programming and Lexical Access

3.1 Introduction

One of the many problems encountered by automatic speech recognisers is the fact that a speaker never pronounces a word twice in the same way. There are always non-linear differences in the durations of two utterances as well as differences in the formant frequencies (vocal tract resonances) and signal amplitudes. A speech recognition system which operates by comparing utterances with stored templates must be able to time align an utterance so that the vectors of parameters representing the same sound in each word correspond with one another.

The concept of dynamic programming (DP) was first introduced by R. Bellman [Bellman 57] in 1957 and was first applied to the problem of time aligning speech signals by T. Vintsyuk [Vintsyuk 68] in 1968. Since then a number of efficient dynamic time warping (DTW) algorithms have been devised which use dynamic programming techniques to find the optimal time alignment of reference templates with isolated utterances [Sakoe & Chiba 78] as well as connected speech [Myers & Rabiner 81] and [Ney 84]. Dynamic programming techniques have also been found to provide a robust and reliable means of lexical access [Thompson *et al* 89]. The apparent similarity between the processes which are involved in dynamic time warping of essentially continuous signals and those which are required to align the naturally discrete phoneme sequences suggest that the efficient DTW algorithms could also be applied to lexical access.

Dynamic time warping (DTW) algorithms are used to time align signals by selectively compressing and expanding portions of essentially continuous signals (although not exclusively speech signals). In practical speech recognition systems the continuous speech signal is sampled and processed to produce a sequence of discrete vectors at regular time intervals (typically every 10mS) and it is these discrete vectors which are time aligned with the corresponding vectors in the reference templates.

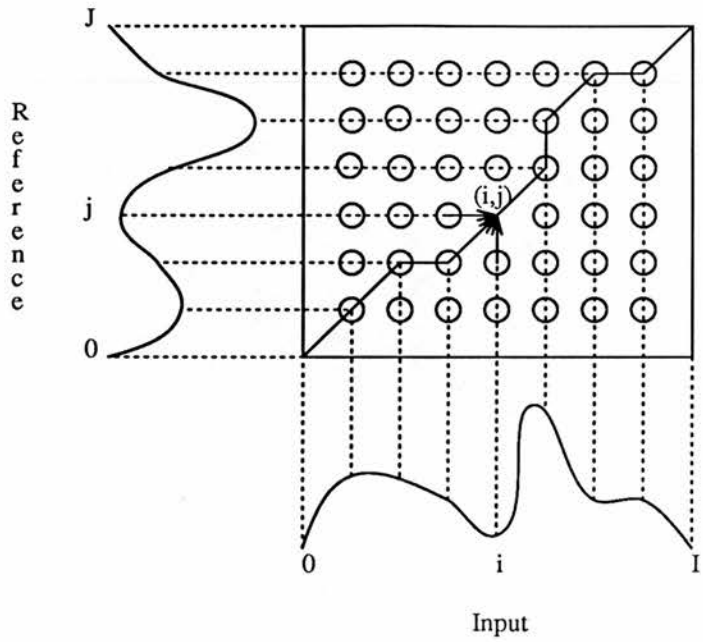


Figure 3–1: A DTW grid showing the path which results in the optimal alignment between the reference and input patterns

The reference template vectors and input vectors define the axis of a set of grid points as shown in fig. 3–1. The problem of finding the optimal time warping is equivalent to finding the path through the grid which minimises the differences between the corresponding frames along each axis as indicated by the dotted lines. Dynamic time warping algorithms use the technique of dynamic programming to gradually construct the optimal time warping between a reference template and a sequence of input vectors by extending intermediate locally optimal alignments. The technique of dynamic programming is based upon Bellman's principle of optimality which can be stated as follows.

'If the best path passes through a grid point (i,j) then the best path includes, as a portion of it, the best path leading up to grid point (i,j)'.

Bellman's principle of optimality allows the optimal (i.e. lowest scoring) time alignment between the reference and input frames to be obtained step by step from the optimal alignments to the preceding grid points. When using this approach the optimal alignment up to grid point (i, j) is simply equal to the optimal alignment leading to the preceding grid point plus the score of the path which is required to reach (i, j) from that point. The globally optimal time warping is obtained by taking the path from the preceding, locally optimal alignment which minimises the accumulated distance score between the template and input frames.

3.2 DTW Isolated Word Recognition

Dynamic programming techniques allow the globally optimal time warping between a template and a sequence of input vectors to be calculated time synchronously from left to right using the locally optimal alignments to the preceding grid points. The set of grid points which may precede a given point are determined by pre-specified path constraints. The path constraints are used to enforce continuity constraints which ensure that the paths always travel from left to right and from bottom to top (i.e. forwards with time, see fig. 3–1) and also limit the degree of local compression and expansion. In this thesis we will be using the symmetrical path constraints (fig 3–4) which allow any grid point (i, j) to be reached from grid points $(i-1, j)$, $(i-1, j-1)$ and $(i, j-1)$. These constraints do not place any limits on the number of consecutive compressions (vertical steps) or expansions (horizontal steps).

The I input vectors and $J(k)$ vectors of each of the K isolated word templates in the lexicon define K matrices of grid points, the first of which is shown in fig. 3–2. Each grid point (k, i, j) identifies an input vector i and a reference vector j in template k and has a local distance score $d(i, j)$ which is proportional to the

difference between vectors i and j. The optimal time warping between a template and the input vectors is obtained by finding the path through the matrix which produces the lowest accumulated distance score $D(k, I, J(k))$ at the end of the alignment. The locally optimal alignment to each grid point (k, i, j) is extended at each time interval i by taking the path which minimises the accumulated distance score $D(k, i, j)$ to that point.

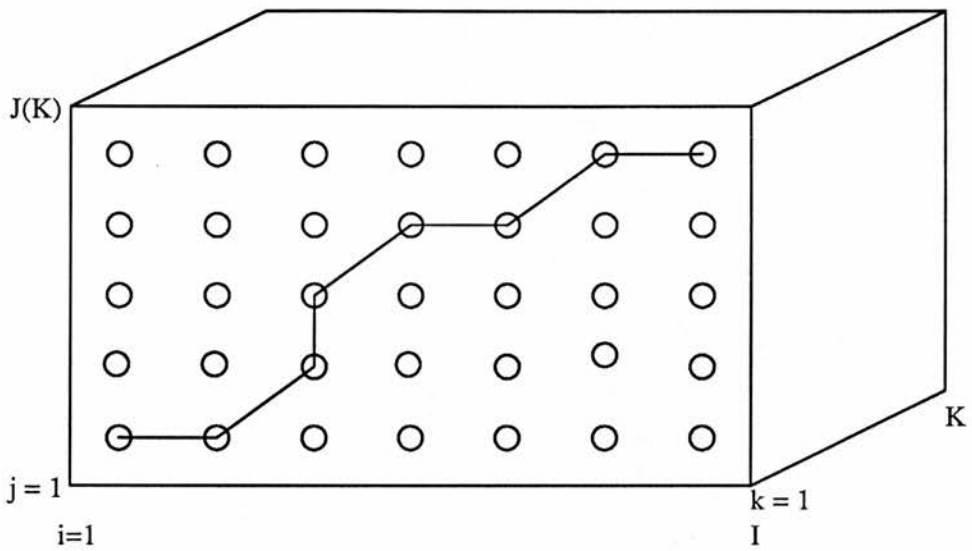


Figure 3–2: Isolated Word Recognition Algorithm: Each of the K isolated word templates are time aligned with the I input vectors derived from the utterance

The locally optimal paths in each template k are propagated time synchronously from left to right as outlined in fig. 3–3. The accumulated distance score $D(k, i, j)$ is calculated for each template and grid point. This score is equal to the lowest value $D^*(k, i, j)$ of the accumulated distance scores which could be obtained by propagating paths from each of the possible preceding grid points. These are calculated by summing the accumulated distance score of the preceding point and product of the local distance score and appropriate time distortion penalty. Once each of the I input vectors have been optimally time aligned with the $J(k)$ vectors of each template k the template with the lowest accumulated distance score $D(k, I, J(k))$ is chosen. The lowest scoring template T is the template which the isolated word DTW algorithm claims to have recognised

Within Word Path Constraints

$$D^*(k,i,j) = \min \begin{cases} D(k,i-1,j) & +d(i,j) * P_e & ; \text{Expansion} \\ D(k,i-1,j-1) & +d(i,j) * P_s & ; \text{Substitution} \\ D(k,i,j-1) & +d(i,j) * P_c & ; \text{Compression} \end{cases}$$

DP Iterations

$$\begin{aligned} &\text{for } i = 1 \ldots I \\ &\quad \text{for } k = 1 \ldots K \\ &\quad\quad \text{for } j = 1 \ldots J(k) \\ &\quad\quad\quad D(k,i,j) = D^*(k,i,j) \\ &T = \text{argmin}_k D(k, I, J(k)) \end{aligned}$$

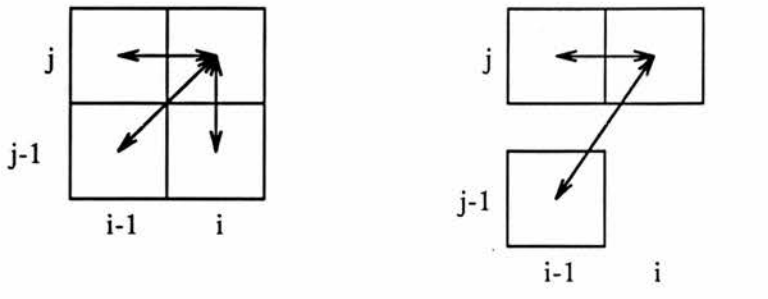
Figure 3–3: DTW Isolated Word Recognition Algorithm

3.3 DTW Connected Word Recognition

The isolated word DTW recognition algorithm can be extended for use in connected word recognition by concatenating the individual isolated word templates to form a super 'reference' template. The super reference template can then be time aligned with the input vectors in an almost identical manner to the templates that were used in isolated word recognition. However, the DTW algorithm now has to find the sequence of reference templates which, when concatenated, give the optimal alignment between the vectors in the templates and the input. Two of the most widely used DTW algorithms for connected word recognition are the level building algorithm [Myers & Rabiner 81] and the one pass algorithm [Vintsyuk 71], [Ney 84] both of which will be briefly described.

Connected word DTW algorithms use different path constraints when propagating paths within templates (fig. 3–4(a))and when propagating paths between templates (fig. 3–4(b)). The between word path constraints are more restrictive than those which are used within words and do not permit new paths to grid point

$(k, i, 1)$ to be propagated from the last grid point $(k, i, J(k))$ of the lowest scoring template k whose alignment also ends at the same time interval i. These path constraints ensure that the first point of a new alignment within a template is always reached by substituting the first frame of the template. These constraints prevent compressions from following expansions (for reasons described in the next section) and also prevent templates from being infinitely compressed i.e. from having alignments which start and end in the same time interval.



(a) Within Word Constraints (b) Between Word Constraints

Figure 3–4: Symmetrical DP path Constraints

3.3.1 The Level Building Algorithm

The level building algorithm sub-divides the connected speech recognition problem into a number of levels each of which basically corresponds to a single application of the isolated word recognition algorithm described previously. Each level is constructed in turn during a separate pass over the input vectors and adds a new template to the super reference pattern. The number of levels is specified in advance and determines the maximum number of word templates that can be present in the final alignment. Figure 3–5 illustrates this process and shows how the super reference template and optimal sentence alignment is constructed by concatenating the optimal alignments of the templates in each of the preceding levels. The algorithm which constructs the super reference template and its optimal alignment is briefly below and in fig. 3–6.

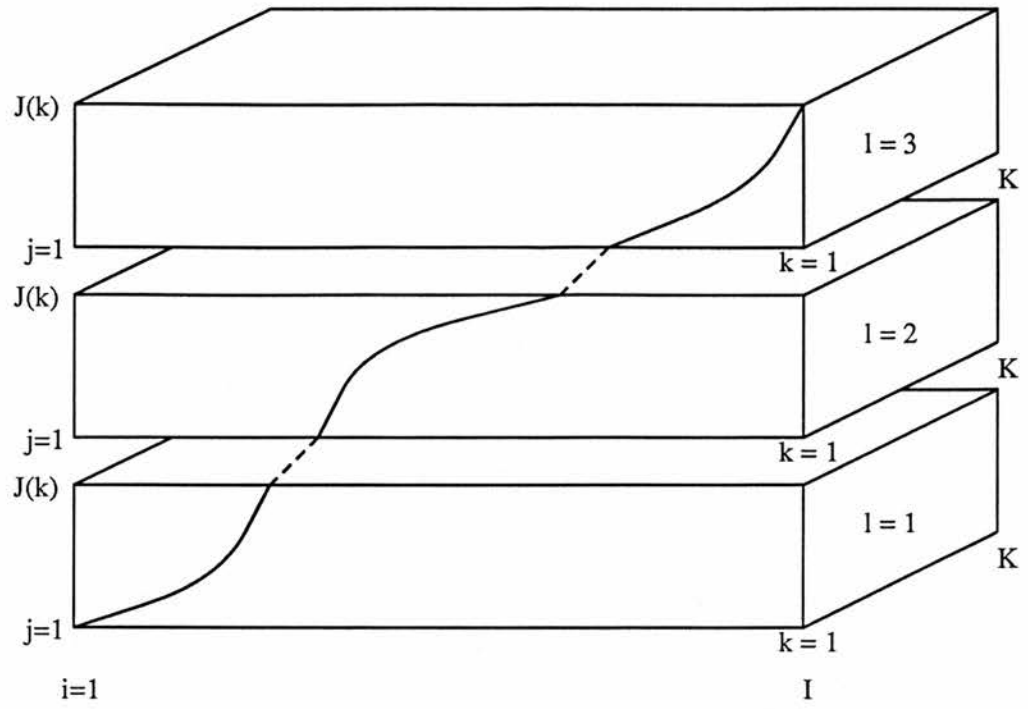


Figure 3–5: The Level Building Algorithm: The super reference template is built up a level at a time during each pass over the input vectors

Each of the L levels is constructed in turn during a separate time synchronous, left to right pass over the input vectors. During each pass the K templates are time aligned with the I input vectors as described previously and at the end of each time interval i the template $T(l,i)$ having the lowest scoring alignment is recorded. New alignments within the templates are propagated using the between word path constraints which allow a new alignment to be propagated from the lowest scoring template $T(l-1,i-1)$ in the previous level as indicated by the dashed lines. The initial accumulated distance score of a new template alignment is equal to the sum of the accumulated distance score $T_D(l-1,i-1)$ of the lowest scoring template in the preceding level and the penalty for substituting the first frame in the new alignment. When the first level is being constructed (i.e. $l=1$) there will be no previous level from which new alignments can be propagated and in this case the accumulated distance score $D(k,i,1)$ of new paths will be zero when $i=1$ and infinity elsewhere. This ensures that the optimal alignment of the super reference pattern always begins at the first input vector. This process

is repeated L times to create L levels where L is the maximum number of word templates that can be concatenated to produce the final alignment.

Within Word Path Constraints

$$D_w^*(k,i,j) = \min \begin{cases} D(k,i-1,j) & +d(i,j)*P_e & ; \text{Expansion} \\ D(k,i-1,j-1) & +d(i,j)*P_s & ; \text{Substitution} \\ D(k,i,j-1) & +d(i,j)*P_c & ; \text{Compression} \end{cases}$$

Between Word Path Constraints

$$D_b^*(k,i,1) = \min \begin{cases} D(k,i-1,1) & +d(i,1)*P_e & ; \text{Expansion} \\ T_D(l-1,i-1) & +d(i,1)*P_s & ; \text{Substitution} \end{cases}$$

DP Iterations

$$\begin{aligned} &\text{for } l = 1 \ldots L \\ &\quad \text{for } i = 1 \ldots I \\ &\quad\quad \text{for } k = 1 \ldots K \\ &\quad\quad\quad D(k,i,1) = D_b^*(k,i,1) \\ &\quad\quad\quad \text{for } j = 2 \ldots J(k) \\ &\quad\quad\quad\quad D(k,i,j) = D_w^*(k,i,j) \\ &\quad\quad T(l,i) = \text{argmin}_k D(k,I,J(k)) \\ &T = \min_{l=1\ldots L} T(l,I) \end{aligned}$$

Figure 3–6: DTW Level Building Algorithm

The optimal super reference pattern T (i.e. the pattern with the lowest alignment score) is obtained by finding the level which contains the template $T(l,I)$ that has the lowest accumulated distance score. The sequence of templates in the optimal super reference pattern and the details of its alignment are obtained by backtracking through the optimal alignment of each of the component templates as indicated by the solid lines in fig. 3–5. The level building algorithm has the advantage that it is also possible to obtain a number of different alignments in addition to the optimal alignment. Alternative alignments containing different numbers of templates can be obtained by backtracking from the last grid point

$(k, I, J(k))$ of the lowest scoring template at the end of the other levels. Backtracking through the lowest scoring template $T(2, I)$ at the end of level two will produce an alignment containing two templates whilst backtracking from level three will produce an alignment containing three templates and so on.

3.3.2 The One-Pass Algorithm

The one pass algorithm builds up the super reference pattern and the optimal alignment during a single pass over the input lattice. This is achieved by effectively collapsing the multiple levels of the level building algorithm into a single level. The one pass algorithm is consequently more efficient in its use of computation and storage than the level building algorithm since only a single pass has to made over the input vectors. There is also no need to place constraints on the maximum number of templates that may be present in the super reference pattern. Figure 3–7 illustrates this process and shows how the optimal alignment of the super reference pattern is constructed from the optimal alignments of the templates in the same 'level'. This algorithm is briefly outlined below and in fig. 3–8.

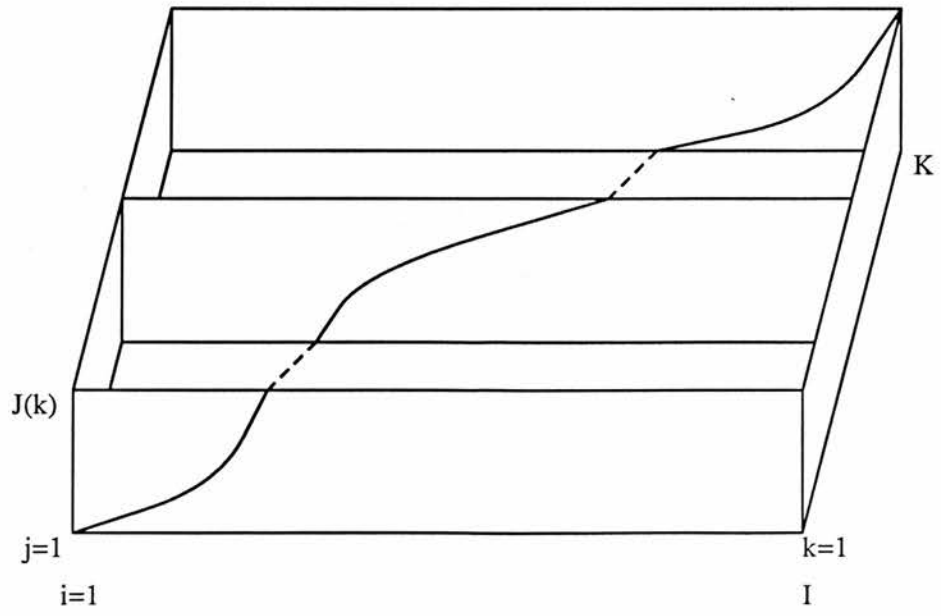


Figure 3–7: The One Pass Algorithm: The super reference template is built up during a single pass over the input vectors

The alignment of the super reference pattern is constructed during a single time synchronous left to right pass over the input vectors. As was the case the level building algorithm each of the K templates are time aligned with the I input vectors and at the end of each time interval i the template $T(i)$ having the lowest scoring alignment is recorded. The between word path constraints now cause new alignments to be propagated from the the lowest scoring template in the previous time interval and the same level (i.e. from $T(i-1)$). As before the initial accumulated distance score of a new template alignment is equal to the sum of the accumulated distance score $T_D(i-1)$ of the lowest scoring template in the preceding time interval and the penalty for substituting the first frame in the new alignment. The accumulated distance score $D(i,1)$ of the initial path is equal to zero when $i = 1$.

Within Word Path Constraints

$$D_w^*(k,i,j) = \min \begin{cases} D(k,i-1,j) & +d(i,j) * P_e & ; \text{Expansion} \\ D(k,i-1,j-1) & +d(i,j) * P_s & ; \text{Substitution} \\ D(k,i,j-1) & +d(i,j) * P_c & ; \text{Compression} \end{cases}$$

Between Word Path Constraints

$$D_b^*(k,i,1) = \min \begin{cases} D(k,i-1,1) & +d(i,1) * P_e & ; \text{Expansion} \\ T_D(i-1) & +d(i,1) * P_s & ; \text{Substitution} \end{cases}$$

DP Iterations

for $i = 1 \ldots I$
  for $k = 1 \ldots K$
    $D(k,i,1) = D_b^*(k,i,1)$
    for $j = 2 \ldots J(k)$
      $D(k,i,j) = D_w^*(k,i,j)$
  $T(i) = \text{argmin}_k D(k,I,J(k))$
$T = T(I)$

Figure 3–8: DTW One Pass Algorithm

The optimal super reference pattern T and the details of its alignment are obtained by backtracking from the last frame $(I, J(k))$ of the template k which has the lowest accumulated distance score at the end of the alignment. The principle disadvantage with the one-pass algorithm is that it is no longer possible to obtain a number of different alignments since alternative points to backtrack from do not exist. However, modifications have been proposed which obtain the N-Best alignments rather than N alignments having different length which are produced by the level building algorithm. These modifications, plus one of our own, are described in more detail in chapter 4.

3.3.3 Summary

The fundamental difference between the level building and one pass algorithms occurs in the propagation of paths between word boundaries. In the level building algorithm paths are propagated from the optimal templates in the preceding level whilst the one pass algorithm propagates paths between templates at the same level. It can be shown that both algorithms are in fact equivalent and will produce identical super reference patterns and template alignments provided that L, the maximum number of levels, is sufficiently large. The following table shows the computational and storage requirements (before any pruning) of the one pass and level building algorithms.

	Computation	Storage
Isolated Word Recognition	$\propto K * J * I$	$\propto K * J$
One Pass Algorithm	$\propto K * J * I$	$\propto K * J + I$
Level Building Algorithm	$\propto L * K * J * I$	$\propto L * (K * J + I)$

where L = maximum number of words (no. of levels)
K = number of templates
J = average template length
I = input length

The computational requirements of the level building algorithm are L times larger than those of one-pass algorithm since the same DP calculations are repeated when constructing each of the L levels. The storage requirements are also slightly larger since the level building algorithm has to record the lowest scoring template at each of the N time intervals when each level is being constructed. The one pass algorithm also has advantages over the level building algorithm in that it does not require a pre-specified maximum number of words that that are allowed to be present in the optimal alignment, negligible additional computational storage expenditure over that of isolated word recognition and ability to operate time synchronously in parallel with a syntactic back end and acoustic front end. We believe that all of these factors will be essential for any large vocabulary continuous speech recognition system.

3.4 Sequence Matching

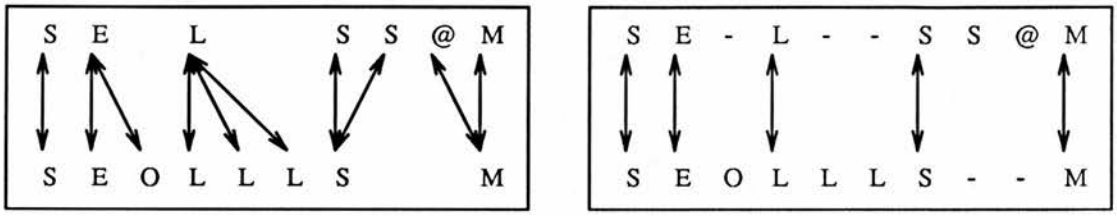
In the preceding sections we have described the principle DTW algorithms that have been used for isolated and connected word recognition. These algorithms operate by selectively compressing and expanding portions of essentially continuous speech signals in order to produce the optimal time alignment between the input signal and the reference template(s). The compression and expansion processes that were used to time align the speech signals are similar to the insertion and deletion processes that would be needed to align naturally discrete sequences such as phoneme transcriptions. The similarities between these two processes suggest that it may be possible to adapt classical DTW algorithms for use in lexical access. In this section we describe the similarities and differences between compressions and expansions, and insertions and deletions.

The basic steps of the dynamic time warping algorithms are compressions, substitutions (when template and input vectors coincide), and expansions whereas the operations which are required to time align naturally discrete sequences are deletions, substitutions, and insertions. It is common to talk about the compression

and expansion of continuous speech signals as being achieved by the deletion or insertion of vectors based upon the following correspondences [Sankoff & Kruskal 83].

Compression-Expansion	Deletion-Insertion
Compress n units into 1	Delete $n-1$ units
Expand 1 unit into n	Insert $n-1$ units

Consider the two alignments of the phonemic transcription of 'sells some' (figure 3–9) one using compression/expansions (fig. 3–9a) and the other using insertion/deletions (fig. 3–9b).



(a) Compression / Expansion

(b) Deletion / Insertion

Figure 3–9: Two Alignments of 'sells some'

Although the alignments produced by the corresponding operations are identical the scores that would be assigned to them could be significantly different. In figure 3–9(a) the reference /l/ phoneme has been expanded to match the 3 surface /l/ phonemes, whilst in figure 3–9(b) the two additional surface /l/ phonemes have been deleted. The penalty associated with the expansion should be related to the similarity of the reference /l/ with each of the surface /l/s. However, if the additional /l/s are deleted then there is no requirement that the two deleted phonemes should have had any similarity with the reference /l/ phoneme in the previous substitution. The score for the expansion will be determined by the similarity of the reference /l/ and each of the surface /l/s plus the time distortion penalties whereas the score for the alternative alignment is simply the sum of the individual penalties for the substitution and two deletions. The dynamic programming algorithm calculates the optimal alignment based upon these local distance scores and since

these can differ it is possible that different 'optimal' alignments will be obtained depending upon whether we use compression/expansions or deletion/insertions.

We should ideally use both compression/expansions and insertion/deletions to find the best alignment. It should be clear from figure 3–9 that the reference /L/ phoneme should be expanded, the two /S/ phonemes should be compressed and the /@/ phoneme should be deleted. The additional /O/ phoneme in the surface string could be either substituted with an expanded /E/ or inserted into the reference string, the correct choice would depend upon the degree of similarity between the surface /O/ phoneme and the preceding reference /E/ phoneme. However, incorporating all of the possibilities increases the number of steps that would have to be considered by a sequence alignment algorithm from 3 (insertion, substitution, and deletion) to 5 (same plus compression and expansion) producing a corresponding increase in computation.

Genuine compressions and expansions are also more difficult to implement since their application is conditional on the preceding step being either a substitution or a matching compression/expansion (fig. 3–9(a)), this would be particularly difficult to ensure when crossing word boundaries. As we have seen DTW algorithms overcome these difficulties by placing further restrictions on the path constraints which only allow the first frame of a new template to be reached via a substitution from the last frame of the preceding template (fig. 3–4(b)) thereby preventing compression/expansions from spanning word boundaries. Unfortunately it is not possible to ignore word initial deletions when aligning phoneme sequences since it is often essential to be able to delete the initial phonemes of words such as 'and' (/a n d/) in phrases like 'fish and chips' (/f i sh n ch i p s/) if realistic alignments are to be obtained.

There are four possible ways that an alignment such as that in fig. 3–9(b) could have been obtained. The alignment could have been obtained by inserting and deleting phonemes in the surface string, inserting and deleting phonemes in the reference string, inserting phonemes in the surface and reference strings, or by deleting phonemes in the surface and reference strings. In theory the choice would appear to be arbitrary and depends upon whether we see the alignment process

as mapping the surface string onto the reference, the reference onto the surface string, or a mixture of both. However the choice can be important since in practice phonemes in the surface string will have likelihood scores assigned to them by an acoustic front end. In this situation it becomes difficult to insert surface phonemes because of the need to provide reasonable likelihood scores and durations which are consistent with those provided by the AFE. The deletion of existing surface phonemes presents no such problems.

The additional computational overhead and implementation difficulties which would arise from the incorporation of compression/expansions into a sequence alignment algorithm have led us to choose to omit compressions and expansions. In the few cases where compression/expansions would appear to be correct choice, due to mis-segmentations by the AFE and phonological assimilations, reasonable alignments can still be obtained with the appropriate insertions and deletions as shown in fig. 3–9(b). Because of the difficulties involved with inserting new surface phonemes we have also chosen to let our lexical access algorithm produce sequence alignments via the exclusive use of surface and reference phoneme deletions. Deletions are achieved by substituting phonemes in the surface and reference strings with a pseudo phoneme '/-/' which has zero duration. Template and input phonemes are therefore always either substituted or deleted. Since input phonemes are deleted rather than inserted into the lattice the problem of calculating scores and durations of inserted phonemes does not arise.

3.5 Lexical Access and Dynamic Programming

The input to our dynamic programming algorithm consists of lattices of phoneme segments which were produced by an acoustic front end developed at CSTR [McInnes *et al* 89]. The acoustic front end segments and labels the acoustic data and for each segment calculates scores for the entire set of phonemes which are equal to the negative log of the probability of their presence given the acoustic evidence. Most of the phonemes in each segment have relatively high scores since

only a few of the 44 phonemes will match well with the acoustic evidence. In practice we only consider a small number of phonemes in each segment that have scores that fall within a pre-specified beam width of the most likely candidate. This leads to approximately six phonemes being considered in each segment.

The phoneme lattices are well-structured in the sense that there are no gaps or overlaps between adjacent segments and all phonemes within a segment have the starting and finishing times. The time t_e at which phonemes in one segment s finish is therefore always equal to the time t_s at which phonemes in the following $s+1$ segment start. This can simplify the lexical access algorithms but is not essential for any of the techniques which we will describe. Researchers at CSTR have experimented with ill-formed lattices containing gaps and overlapping segments but experience showed that with improvements in the front end and the use of dynamic programming for lexical access it was reasonable to demand a connected and uniquely segmented lattice.

Each individual reference template contains a string of phoneme symbols which describe a single citation form pronunciation of the word. Grammatical constraints can be used to restrict the ordering of templates in the super template to those which are syntactically valid. In order to facilitate this we have partitioned the lexicon according to the $c = 1 \ldots C$ terminal categories of the grammar (App. A.2) such as noun, verb etc.

The chart parser[1] can be used at word boundaries to ensure that alignments are only propagated between templates in categories which are syntactically valid. Grammars are usually implemented in the form of augmented transition networks or finite state networks which are efficient but inflexible. In contrast we use a chart parser which processes the phrase structure grammar rules directly. Although this is less efficient it is flexible and as we shall see much of the computation can proceed in parallel with the DP lexical access algorithm.

---

[1]The chart parser is described in more detail in Chapter 5.

It has been shown [Godin & Lockwood 89], that in the presence of syntactic constraints, the one-pass and level-building DTW algorithms are functionally identical and will produce the same alignments. However, this only appears to be true when they are used for dynamic time warping where the more restrictive path constraints are used at template boundaries which do not allow word initial deletions. As we have seen word initial deletions can be essential in order to obtain the optimal alignments during lexical access. The one-pass algorithm runs into problems with word initial deletions which would not encountered by the level building algorithm.

In the following section we will briefly describe a general time synchronous algorithm which can be used to implement either of these two algorithms depending upon how paths are propagated at template boundaries. We will then describe how dynamic level building is used to overcome the difficulties that arise from word initial deletions whilst preserving the relative advantages of the one-pass algorithm.

3.5.1 The DP Algorithm

The DP algorithm outlined in fig. 3–10 and described below uses Bellman's principle of optimality to construct the optimal alignment a step at a time. It can be used to implement either a one pass or level building DP algorithm, with or without syntactic constraints. Symmetrical path constraints are used (fig. 3–4(a)) which permit an unlimited number of consecutive insertions and deletions. Unlike DTW algorithms we do not distinguish between within word and between word path constraints.

The input lattices which are consumed by the DP algorithm consist of S phoneme segments each of which contains I different phonemes. Each phoneme i in a segment s has the same starting time $t_s(s)$ and the same finishing time $t_e(s)$. The phonemes in each segment are ranked according to a likelihood measure $ac(i)$ of their presence which is calculated by the AFE. The lexicon is partioned into C syntactic categories according to the terminal categories of the grammar. Each

category c in the lexicon contains $K(c)$ templates and each template k contains $J(k)$ frames which contain the phonemic labels for a single citation form pronunciation.

$$D^*(c,k,s,j) = \min_{i=1\ldots I(s)} \begin{cases} D(c,k,t_s(s),j) & +d(i,-) & +ac(i) & \text{;Delete input} \\ D(c,k,t_s(s),j-1) & +d(i,j) & +ac(i) & \text{;Substitute} \\ D(c,k,t_e(s),j-1) & +d(-,j) & & \text{;Delete template} \end{cases}$$

for $l = 1 \ldots L$

for $s = 1 \ldots S$

1) Between Word Syntactic Constraints

for $c = 1 \ldots C$

for $k = 1 \ldots K(c)$

$D(c,k,s,0) = \min_{c_s=1\ldots C_s} T_i(t,c_s)$

2) Within Word DP Alignment

for $c = 1 \ldots C$

for $k = 1 \ldots K(c)$

for $j = 1 \ldots J(k)$

$D(c,k,s,j) = D^*(c,k,s,j)$

$T_i(s,c) = \text{argmin}\,\{D(c,k,s,J(k))\}$

Figure 3–10: The Unified DP Alignment Algorithm

For every template k in each of the categories c, the accumulated distance score $D(c,k,s,0)$ of the new path is initialised using values from the lowest scoring template $T_i(t,c_s)$ chosen from the categories c_s that are syntactically valid predecessors of c. In the case where there are no syntactic constraints any category can precede any another so the set of valid predecessor categories c_s would

equal the entire set of categories C. When a parser is being used to restrict the relative ordering of categories c_s would typically be a small subset of C. If there are no syntactically valid categories no new paths will be initialised and extended in the subsequent stages of the DP algorithm.

The dynamic programming algorithm gradually extends the optimal alignment by finding the lowest scoring extension $D^*(c,k,s,j)$ from each of the preceding paths as determined by the symmetrical path constraints (fig. 3–4(a)). The accumulated distance score $D(c,k,s,j)$ of the new path is calculated by taking the minimum value of the sum of the previous accumulated distance score plus the substitution penalty $d(i,j)$ and likelihood score $ac(i)$ of the substituted input phoneme. When all of the templates in a category c have been processed the template $T_i(c,s)$ with the lowest alignment score is selected and recorded.

Backtracking pointers (not shown) are also computed and stored so that the optimal path can be displayed once the alignment is complete. When between word transitions occur pointers to the preceding template are stored which allows the optimal sequence of templates to be retrieved by backtracking through the list of pointers. If the individual phoneme substitutions along the optimal alignment are also required than each grid point will also have to contain back-pointers to the preceding point from which the path was extended.

The source of the lowest scoring templates $T_i(t,c(s))$ along with the range of the index l determines whether a one pass or level building algorithm is implemented. In the level building algorithm the set of templates $T_i(t,c(s))$ will be retrieved from the previous level $(l-1)$ (fig. 3–11(a)) whereas in the one-pass algorithm the templates are retrieved from the previous segment in the same level (fig. 3–11(b)). This choice does not depended upon whether or not syntactic constraints are employed. By calling functions which either retrieve preceding templates from the previous or the same level we can use the same body of code for training (level-building algorithm) and evaluation (one-pass algorithm) with or without an optional parser.

If we are not interested in observing the sequence of phoneme substitutions along the optimal alignment then significant savings in storage requirements can

be achieved. From fig 3–4 we can see that each path can only be extended from the paths calculated in the preceding segment. Since there is no need to refer back further than the preceding segment when extending paths, paths which occur before this can be discarded. However, if we do not assume that the lattice is well-structured then we will have to keep sufficient segments to ensure that the all of the paths located at the starting time $t_s(i)$ of each phoneme can still be found. In practice this is achieved by setting two beam values, the first determines the duration over which the most recent segments need to be preserved and should be greater than the maximum duration of any phoneme. The second parameter determines how often pruning takes place and determines the balance between reduced storage and increased computational overheads due to pruning. For example with values of 300mS and 200mS, every 200mS paths further back than 300mS from the end of the current segment time will be pruned.

Coping with Word Initial Deletions

In traditional DTW algorithms the between word path constraints (fig. 3–4(b)) are more restrictive than the within word path constraints (fig. 3–4(a)) which do not allow word initial deletions. We have not distinguished between these two cases since we need to have word initial deletions so that alignments such as (@.-)(n.n)(d.-) for 'and' and (dh.-)(@.@) for 'the' are possible. The introduction of word initial deletions results in a number of problems for the one-pass DP algorithm which would not be encountered by a level building algorithm. The reasons for the difficulties can be seen by examining the way in which paths are extended across word boundaries in the level building and one pass algorithms as indicated by the arrows in fig. 3–11.

The level building algorithm constructs the optimal template alignment level by level. The optimal templates in one level are determined before work begins on the next level. The paths which are obtained via word initial deletions (corresponding to the vertical arrows in fig. 3–11) are extended from the paths in the final frame of templates in the preceding level. The word initial paths in the new template are therefore extended from paths in the preceding level that finish *at the same time*

as the new paths originate. This is not a problem for the level-building algorithm since the best scoring templates containing paths which finish at this time will already have been determined and will be recorded in the previous level.

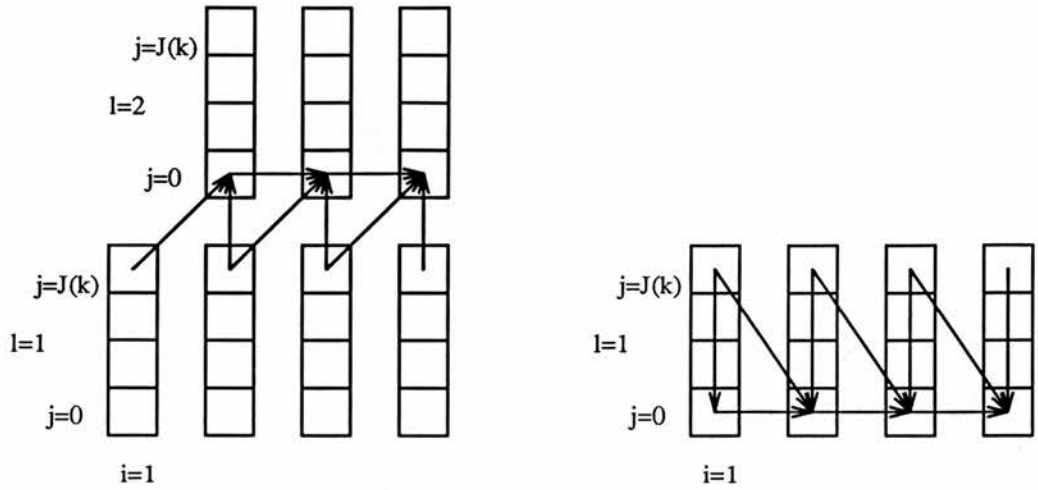


Figure 3–11: Propagating paths between templates in the two DP algorithms

The one-pass algorithm is computationally more efficient since it effectively collapses the numerous levels of the level-building algorithm into a single level (fig. 3–11(b)). The paths which are obtained via word-initial deletions (vertical arrows) are now extended from the best scoring templates in the same level. This leads to a paradox since we cannot determine the lowest scoring template alignment until we have calculated scores for each of the $j = 0 \ldots J(k)$ reference frames. However we cannot calculate score for the initial frame $j = 0$ and subsequent frames until we know the score of the optimal template since the optimal first substitution of the alignment may consist of a word initial deletion from this template.

The solution to breaking this circle is to observe that the path scores increase monotonically since the individual substitution penalties are by definition positive (see Ch. 3.5.2). Because of this we can determine the optimal template alignments in two passes. In the first pass word initial deletions are not used and the optimal alignments are propagated using the restrictive between word path constraints in

fig. 3–4(b). Once the best scoring template has been determined we then go back and make a second pass which implements the missing word initial deletions. If the optimal path to the first frame should have been a word-initial deletion the old 'incorrect' path is replaced by the deletion. When this occurs the path to the next frame $j + 1$ will also have to be reconsidered since the optimal path to this frame may now consist of a deletion from the preceding frame j which has just been modified. This process will have to be repeated for the remaining frames $j = 1 \ldots J(k)$ until the score of the prospective deletion is higher than the score of the existing path. This procedure is outlined below.

for $c = 1 \ldots C$
  for $k = 1 \ldots K(c)$
    while $j < J(k)$ and $D(k,i,j) > D(k,i,j-1) + d(-,j)$
      $D(k,i,j) = D(k,i,j-1) + d(-,j)$

Figure 3–12: The second pass of the DP alignment algorithm fills in the missing word initial deletions

This second pass will never affect the initial choice of the lowest scoring template which would in turn invalidate the scores of the existing word initial deletions. If all of the reference phonemes in a template were to be subsequently deleted the score of the path would be equal to the score of the previously determined optimal template (from which the path originated) plus the sum of the individual deletion penalties. Since the penalties are always positive the alignment scores increase monotonically. The new alignment score of a completely deleted template therefore can not be lower than the score of the previously determined optimal alignment. This also means that, in the absence of grammatical constraints, a completely deleted template will never be part of the optimal alignment since there will always be at least one other template that has a lower alignment score.

3.5.2 Dynamic Level Building

The two pass approach described previously would be sufficient for the simplest cases where the DP algorithm generates a single optimal alignment which is not subject to grammatical constraints. Unfortunately, when we come to generate the N-Best alignments (Ch. 4.2), or enforce grammatical constraints (Ch. 5.2) it is possible for a completely deleted template to form part of an N-Best or syntactically constrained alignment. When this occurs we will we have to update our initial choice for the set c_s of lowest scoring templates and this means that we will also have to go back and recompute word initial deletions which originated from templates which may no longer be among the initial set or which may have lower scores if they were to originate from one of the newly deleted templates. This may in turn cause further changes to set of N-best or syntactically valid templates.

The solution to this more complex problem is an extension of that described in the previous section. Word initial deletions are still applied after the main DP pass but now the scores of templates that have had their reference frames completely deleted are compared against the initial choice of the lowest scoring templates. If any of the newly deleted templates have a lower score than any of those in the initial set a new 'level' of backtracking information is created which records the backtracking pointers for the newly deleted templates. The second word initial deletion pass is then repeated using deletions which extend paths from the final frames of the newly deleted templates rather than from the final frames of templates in the initial set.

```
REPEAT
    Apply between word syntax constraints
    Apply word initial deletions
    IF a deleted template has a lower score THEN
        store backtracking data
UNTIL no new templates have lower scores
```

Figure 3–13: The dynamic level building algorithm

A number of levels of template backtracking information will have to be stored, one for each set of templates that have alignments which finish at the same segment (fig. 3–14). Each successful iteration of dynamic level-building will result in the creation of a new level of backtracking information. The backtracking pointers for these completely deleted templates will refer to information about templates in the preceding level whose optimal alignments also ended at the same time interval. The dynamic level building step is always executed once to allow for an arbitrary number of word initial deletions and is then repeated indefinitely whilst the process results in creation of new, lower scoring templates. In practice it is rare for even one template to be completely deleted.

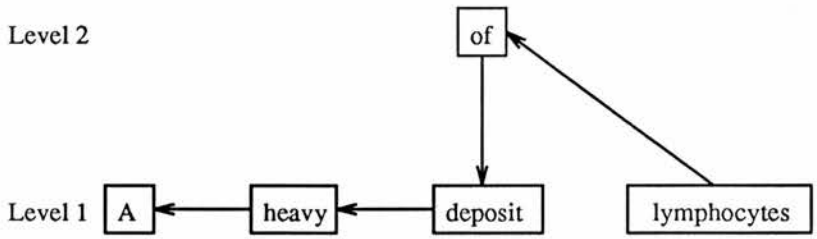


Figure 3–14: Backtracking pointers stored during dynamic level building

Although separate instances of template backpointers are recorded in different levels of backtracking information, the paths in the template frames are overwritten during the application of word initial deletions. This means that the original phoneme level backtracking pointers are lost when the phonemes in a template are deleted during the second pass . Problems will arise if the same word template appears consecutively twice or more in the optimal alignment with the template phonemes being completely deleted in the latter occurrence(s).

If the same word template appears consecutively in the optimal alignment the dynamic level building stage will overwrite the original paths and phoneme back-pointers with deletions. Although it would be possible to backtrack through the phoneme substitutions in the last, completely deleted template, it would not be possible to backtrack through the substitutions of the previous occurrence of

the template since the necessary pointers will have been overwritten. Fortunately the likelihood of obtaining two consecutive alignments of the same template in an optimal alignment are very small, especially when syntactic constraints are in force and we have never encountered this problem in practice.

The dynamic level building preserves the relative advantages of the one pass algorithm over the alternative level building algorithm. The additional levels of backtracking information are dynamically created as and when they are required so computational and storage overheads are kept to a minimum. There is also no need to specify a maximum number of levels. The second word initial deletion stage is also much quicker than the first DP stage since the only substitutions which are used are template deletions whereas the first DP stage must consider template deletions, template and input substitutions and input deletions.

Maximum Likelihood (ML) Training

The dynamic programming algorithm obtains the optimal (lowest scoring) alignment on the basis of the scores provided by the acoustic front end and a matrix $d(i,j)$ of substitution penalties which are trained using an iterative maximum likelihood training procedure. The cost of substituting a reference phoneme i with a surface phoneme j is equal to the likelihood score assigned to the phoneme by the AFE plus a constant pre-determined substitution penalty $d(i,j)$. The substitution penalties indicate the inherent confusabilities between the lexical and surface phonemes and are calculated as scaled negative log probabilities using the re-estimation formula in eqn. 3–15.

$$d(reference, surface) = -128 * \log_2 \frac{t(lexical.surface)}{\sum_{p=0...P} t(p, surface)}$$

Figure 3–15: ML Re-estimation Formula

The substitution penalties are recursively re-estimated using a maximum likelihood (ML) training procedure which attempts to maximise the likelihood of the alignments between a set of training sentences and the corresponding phoneme

lattices produced by the acoustic front end. During training, the system is run with sequence of templates constrained so that the correct sentence is always obtained for each utterance in the training set. The optimal alignment is obtained for each training sentence and a matrix of tallies (app. A.3) is updated which records the number of times each of the reference and surface phoneme were substituted. The tallies $t(lexical, surface)$ are used to re-estimate the substitution probabilities from which the scaled negative log substitution penalties are calculated. This process is then repeated using the updated substitution penalties and continues until the total accumulated distance score summed over all training sentences converges onto a (local) minimum. This typically takes an average of six passes over the training corpus.

The ML training procedure is only guaranteed to find a local minimum in the total accumulated distance score. It is therefore important to provide a realistic set of initial substitution penalties in order that a good (and hopefully globally optimal) solution will be found. Our initial matrix was constructed from simple linguistic motivations which determined the confusability of phonemes by comparing basic phonetic features such as voicing, manner, and place of articulation. The penalties for feature mis-matches were ranked in descending order of voicing, manner, and place. The initial substitution penalties are then simply calculated as the sum of the penalties associated with individual feature mis-matches between the two phonemes.

3.6 Base-line Results

The following results were obtained from the dynamic level building algorithm using a set of 80 sentences and pre-computed lattices which were provided by CSTR along with a lexicon containing 206 word templates. Each template contains a single phonemic transcription of the citation form utterance of the word. The substitution penalties were trained using 60 of the sentences which provided an average of 2400 phoneme substitutions. The remaining 20 sentences, which had

not been seen before, were used to perform an open test. This procedure was then jack-knifed by taking 4 different partitions of 60 training and 20 evaluation sentences. We also performed a closed test by training and evaluating on the same set of sentences.

A simple modification [McKelvie 90] was made to the lexical access DP alignment algorithm. This modification involves incorporating a constant word penalty into the scoring calculations of the DP algorithm described in the previous section. The word penalty is added to the start of each new alignment of a template so the cumulative effect of the penalty is therefore equal to the number of templates in the final alignment. As a consequence alignments containing fewer templates will be propagated by the DP algorithm in preference to alignments having more templates which will accumulate more of the word penalties. This has the effect of eliminating a large number of occurrences of short words such as 'a', 'the', 'of' which contain a small number phonemes that can be easily aligned with the lattice segments. Although this also has the unfortunate effects of eliminating the correct occurrences of these words the benefits which are gained by eliminating most fortuitous matches more than outweighs the losses.

A value was chosen for the word penalty offset which appeared to give the best compromise between the aims of eliminating fortuitous matching whilst retaining correct matches. The value which was finally chosen is roughly equivalent to half the average phoneme substitution penalty. Using this word penalty almost doubled the word recognition rate from 31% to 60%. The basic recognition rates using this word penalty are listed below.

	Open Test	Closed Test
Word level recognition	60%	63%
Sentence level recognition	6%	7%

One reason for the low sentence recognition rate was the poor recognition of short function words such as 'the', 'of' and 'a' which were frequently omitted because of the addition of the word penalties. The following alignment which was obtained is correct apart from the missing (bracketed) function words.

```
(the) bronchial washings contain (a) few clumps (of) malignant cells
```

If we were to overlook the missing function words the sentence level recognition rate would increase to 28%. The poor recognition of function words is a well-known problem for continuous speech recognisers and as a result these words are often treated as special cases which are described by specialised phoneme models or additional phonemic pronunciations [Lee 88]. In later chapters we will show how macro substitutions are able to capture the difficulties with which some function words are recognised and are able to provide some help by generating alternative pronunciations at run time.

The actual phoneme alignments such as those listed in the appendix also highlight other limitations of the acoustic front end. Diphthongs such as /i@/ and long phonemes such as /l/ were often mis-segmented. In the case of /i@/ this resulted in the two individual phonemes /i/ and /@/ being aligned instead of the single phoneme /i@/. Many other non-identity substitutions also occurred, even when the 'correct' phoneme was known to be present in the lattice segment. These could have been caused by a combination of over training and the presence of multiple phoneme hypotheses in each segment. For example, whenever the AFE hypothesises a /p/ it will usually also hypothesise a /d/ with a similar score. If, during training the /p/ is often confused with /d/ the penalty for the substitution (p.d) will be relatively low, possibly lower than the identity substitution (p.p). As a result the lexical access algorithm will prefer to perform the substitution (p.d) even though the 'correct' /p/ phoneme is present in the lattice. Of course, other non-identity substitutions will be caused by the absence of the correct phoneme and occasionally unreasonable substitutions are encountered such as (g.m). These are likely to be due to the limited amount of training data which was available to estimate the substitution penalties. The lack of training data would also account for the small improvement in recognition accuracies that were observed in the closed test.

We have observed two cases where expansions and compressions would have been useful to cater for word boundary assimilations and the mis-segmentation of

long phonemes. Word boundary assimilations occurred when two similar phonemes merged across a word boundary (for example /s/ in /s k w ei m @ s e l z/ 'squamous cells'). Mis-segmentation of phonemes, particularly /l/ are caused by deficiencies in the front end which cause a single utterance of the phoneme /l/ to be mis-segmented and recognised as a sequence of two or more individual /l/ phonemes. The additional /l/s are always deleted rather than being compressed during the training process. The probability of deleting a /l/ during evaluation is therefore relatively high. One consequence of this is that it too easy to delete /l/s in the input lattice during evaluation and this increases the likelihood of obtaining fortuitous matches. This problem will become more noticeable when we group sequences of deletions in the form of macro-substitutions.

Mis-segmentations and non-identity substitutions also cause correct templates to be produced with poor alignments. The template 'cells' was often poorly aligned with a high score but the word is still reliably detected whereas the template 'rapid' had a reasonable alignment and relatively low score but should not have been recognised.

```
'cells'        (s.s)(-.@)(e.o)(-.l)(-.l)(l.l)(-.z)(z.z)
'rapid'        (r.r)(a.a)(-.ng)(p.d)(i.i)(d.d)
```

This example shows that the quality of the alignment does not always reflect the correctness of the template and would appear to indicate that there is limited scope for beam width pruning since the alignment of some correct words may have much higher scores than the alignments of other incorrect words.

The ability to study the phoneme confusion matrices which are generated during training (appendix A.3 have been found to be an invaluable tool in the development of speech recognition algorithms [Picone *et al* 86], [C Bourjot & Fohr 89] and [Browning 91]. They can be used to highlight areas of weakness in the AFE and show where attention needs to be focused in order to produce the largest increases in phoneme recognition performance. However confusion matrices provide no information about the context in which substitution errors occur. This

means that it impossible to identify context sensitive errors and distinguish between deletion of genuinely erroneous phonemes and those which occur as a result of mis-segmentations. Many of these limitations are overcome by the correlation trees from which our macro-substitutions are extracted.

Chapter 4

Enhancements and Extensions

In this chapter we will spend a little time describing the extensions that have been made to the DP lexical access algorithm described in the previous chapter. The first of these is a technique for the corrective training of substitution penalties which aims to reduce the number of statistically significant differences between the frequencies of phoneme substitutions observed during training and those observed during evaluation. In the next section we describe an algorithm which efficiently generates over 99% of the genuine ten best optimal paths whilst requiring 80% less storage and computation than the original N-best algorithm. In the last section we present the results of an investigation into the extent to which beam-width pruning could be used to further reduce the size of the search space which shows that there is an optimal range for the beam-width threshold.

4.1 Corrective Training

The ML training procedure described in chapter 3.5.2 maximises the probability of the alignment between the phonemes which make up the training sentences and the phoneme lattice. The sequence of templates for each training sentence is known in advance and plays no part in the training process other than to define the sequence of phonemes which make up each training sentence. The training process can therefore be regarded as one of optimising the probability of the alignment of

an independent string of phonemes with the lattice. ML training ignores the fact that phonemes are organised in groups defined by the sequence of word templates and as a result the independent probabilities (and hence the substitution penalties which are derived from the probabilities) that are calculated during training are incompatible with the actual substitution probabilities that will be observed during evaluation.

During evaluation the input lattice is also aligned against a sequence of template phonemes but the individual phoneme substitutions are no longer independent. The lexical templates constrain the possible sequences of phoneme substitutions so the occurrence of one particular substitution will be dependent upon the presence (or absence) of previous and following substitutions. Consequently, the independent substitution probabilities that were calculated during ML training do not have a straightforward interpretation as probabilities when used during evaluation. If ML training yielded a probability of 0.05 for a particular substitution during training there seems little reason to expect that by using this probability we would also observe 0.05% of these substitutions during evaluation. The probability will depend not only on its own substitution penalty, but also on the penalties of all the other substitutions within the same template in which it occurs.

For example, multiple occurrences of the alignment (i.i)(z.s) of the word 'is' in the training sentences may lead the ML training procedure to calculate a probability of 0.02 for the substitution (z.s). The probability of the substitution (i.i) is likely to be much higher since this substitution would have also occurred in the alignments of many other words such as 'this' and 'consists'. We should not be surprised if the probability of observing the substitution (z.s) was much higher than 0.02 during evaluation due to the presence of fortuitous matches with phoneme sequences that might have originated from the pronunciations of words like 'this' and 'consists'.

The frequency of fortuitous matches will also be increased due to the presence of multiple phonemes within each segment of the lattice. The relatively high probability of the substitution (i.i) means that this substitution is likely to occur whenever there is an /i/ phoneme in the segment, even if it is not the lowest

scoring phoneme hypothesis in the lattice segment. The high probability of the substitution (i.i) will increase the probability of 'recognising' the template 'is' which in turn increases the probability of observing the substitution (z.s) since both of these substitutions occur together in the alignments (i.i)(z.s) of 'is'.

Ideally, we would like the probability of phoneme substitutions observed during training and evaluation to be identical. If the probabilities of observing the individual substitutions during training and evaluation are similar then the alignments which result from these substitutions are also likely to be similar. Intuitively, we would also expect the recognition rates to be higher when the alignments are similar.

However, it is often the case that the correct sequence of templates is not obtained and the phoneme alignments of the templates will almost certainly be incorrect as well. When the alignments differ the tallies of the phoneme substitutions will also differ according to whether the substitutions occurred too often or too rarely. The corrective training procedure which we propose involves modifying the substitution penalties in such a way as to reduce the differences between the probabilities of observing each phoneme substitution during training and evaluation.

4.1.1 Methodology

The aim of corrective training is to reduce the number of statistically significant differences between the frequencies of phoneme substitutions observed during training and evaluation. This will involve decreasing the substitution penalties of those phoneme substitutions which occurred more often during training and likewise, increasing the substitution penalties of those phoneme substitutions which occurred more often during evaluation. The magnitude of the modification should be related to the relative differences between the substitution probabilities.

A new estimate $d(i,j)'$ of the penalty of a substitution (i,j) could be calculated as shown below by multiplying the old penalty $d(i,j)$ by the relative probability of observing the substitution during training $P_t(i,j)$ versus the probability $P_e(i,j)$

of observing the same substitution during evaluation. The following equation will modify the substitution penalties in the manner described above.

$$d(i,j)' = d(i,j) * P$$

where

$$P = \frac{P_t(i,j)}{P_e(i,j)}$$

Experience has shown that such a simple approach is unsatisfactory. The relationship between the substitution penalties $d(i,j)$ and the observation probabilities $P_e(i,j)$ was found to be complex and highly non-linear. Small changes in some substitution penalties would have a disproportionately large effect on the observation frequencies whilst some large changes would have negligible effects on the observation frequencies of other substitutions.

This problem was overcome by introducing a damping factor D which enables the extent to which the substitution penalties are modified due to differences in the observation frequencies to be gradually reduced. The damping factor is initially set to 0.0 which allows the full correction factor P to be applied to quickly correct those penalties which require large changes. Once these have been corrected the damping factor is gradually increased towards 1.0 thereby scaling down the range of P in order to allow the more sensitive substitution penalties to be corrected.

$$P = 1.0 + \left(\frac{P_t(i,j)}{P_e(i,j)} - 1.0\right) * (1.0 - D)$$

It is first necessary to center the relative substitution probability $\frac{P_t(a,b)}{P_e(a,b)}$ around zero. This is needed to ensure that the multiplication by the scaling factor $(1.0-D)$ only affects the magnitude and not the direction of the correction factor. Once the multiplication has been made the correction factor is once again recentered around 1.0. When the damping factor D is zero the equation is equivalent to $\frac{P_t(a,b)}{P_e(a,b)}$ but as the damping factor increases the significance of any deviation of the relative substitution probability from its ideal value of 1.0 will be reduced. This simplifies to the following equation :-

$$P = \frac{P_t(i,j)}{P_e(i,j)} * (1.0 - D) + D$$

The absolute difference between the two probabilities P_t and P_e is used to calculate a confidence measure C which is expressed in units of the greatest uncertainty $\delta P(i,j)$ in the two probabilities. This measure will have a value of 1.0 when the difference between the two probabilities is equal to the largest estimate of the uncertainty in either probability, a value of 2.0 when the difference is twice as large as the maximum uncertainty and so on. The confidence measure provides an indication of the significance of the differences between the two probability estimates. If the difference is to be considered reliable it should be much larger than either of the uncertainties in the probabilities.

$$C = \frac{|P_t(i,j) - P_e(i,j)|}{\max(\delta P_t(i,j), \delta P_e(i,j))}$$

The confidence measure C is used to eliminate statistically insignificant differences by ignoring all differences which have a confidence measure less than some predefined limit. An initial value of two will ensure that the differences are not simply products of the uncertainties of the individual probabilities. The remaining statistically significant differences are used to calculate the modification factors which are then used to produce an updated estimate $P(i,j)$ of the substitution probabilities. The corrected penalty matrix is used to regenerate the sequences of phoneme substitutions which are produced during evaluation and a new set of differences are obtained. As corrective training proceeds recursively the damping factor and confidence limit are gradually raised until no significant differences remain or a predefined limit on the number of passes is exceeded.

4.1.2 Results and Discussion

The cytology corpus was used which consists of 80 sentences and their associated phoneme lattices. Of these 60 were used to generate the phoneme sequences for ML and corrective training and the remaining 20 were used to calculate the word

recognition scores for evaluation. The probabilities and estimates of their uncertainties were gathered and calculated by Treelab [Rohwer 87], a statistical N-gram modelling package (this package is described in more detail in Ch. 6.2). Ideally we would have compared the 60 training and 20 evaluation alignments when looking for significant differences. Unfortunately, the use of only 20 alignments would not result in sufficient data from which reliable and representative probabilities could be calculated.

The 60 training sentences were used to calculate the optimal alignments using both the training algorithm which aligns the phoneme lattice with the correct sequence of templates and the evaluation algorithm which determines the optimal sequence of templates to align with the phoneme lattice. The two sets of alignments which were obtained from the first corrective training pass produced a total of 2378 and 2253 individual phoneme substitutions from the training and evaluation alignments.

These alignments were then processed by Treelab and ten significant differences between the frequency of occurrence of phoneme substitutions in the training and evaluation alignments were detected and used to correct the substitution penalties as described. This process was then repeated using with the corrected substitution penalties. Subsequent use of the new penalties caused many fortuitous matches of short words to be reintroduced into the alignments. The word penalty had to be increased by more than 50% before these were once again eliminated but this only happened after the first corrections had been made and the word penalty did not have to be changed on subsequent passes.

Several passes of corrective training were found to be necessary as the relationship between substitution penalties and the frequency of observation is not straightforward. The relationship is non-linear so small changes in the penalties can produce large changes in the frequency of occurrence, whilst some large penalty changes can have little effect. Furthermore, the absolute value of a substitution penalty is not so important as its value in relation to all other penalties in the matrix (e.g. dividing all of the substitution penalties by 2.0 would have no effect on the alignments obtained during evaluation other than reducing the accumulated

distance score). Therefore, changing one substitution penalty will often effect the observation probabilities of many other substitutions whose penalties may then have to be corrected on subsequent passes.

In extreme cases this led to oscillations which occurred when the frequency of occurrence was particularly sensitive to changes in the penalty score. In these cases reduction of the penalty would lead to the substitution occurring too frequently and then after the following pass when the penalty was increased the substitution would once again occur too infrequently. This would proceed indefinitely if the damping factor D was not available to gradually scales down the magnitude of the changes in the substitution penalties in order to force the convergence of oscillating penalties onto a reasonable value as the corrective training procedure progresses.

The corrective training procedure was jack-knifed using four different partitions of 60 training and 20 evaluation sentences and was found to produce a small but consistent increase in word recognition scores of 6 to 10 percentage points. Although only statistics for 1-grams (i.e. single substitutions) were used, it should be possible to extend these techniques to higher order N-grams thereby further increasing the statistical similarity of the trees resulting from the training and evaluation sequences and hopefully yielding further increases in recognition accuracy. It is also interesting to note that although corrective training could result in negative substitution penalties although this has not appeared to cause any problems so far.

4.2 The N-best Algorithm

It is desirable for the lexical access module to provide a number of possible alignments since this increases the probability of finding the correct alignment. The full N-best algorithm [Schwartz & Chow 87] provides this capability but this results in large increases in computational and storage requirements. Sub-optimal variants have been proposed [Steinbiss 89] and [Marino & Monte 90] which generate a number of plausible (though not necessarily optimal) alternatives with much

less work. We have proposed an extension to the N-best algorithm [Nowell 91] which, when applied to our problem of lexical access, reduces the computational and storage requirements by 80% whilst still calculating ten paths with more than 99% probability that these are the genuine ten best optimal paths. Below we will outline the general principles of the N-best algorithm and then show how a simple extension can be used to substantially reduce the amount of work that needs to be done.

The N best alignments can be obtained by a straightforward extension of the DP algorithm described in Ch. 3.5.1 which extends the two dimensional path searching problem into three dimensions, the third dimension being the number of paths (N) that are calculated at each point in the matrix. Bellman's principle of optimality can be reformulated for the 3-dimensional case as follows:

> If, for $n = 1, \ldots, N$ the n-th best path passes through some intermediate state (i, j), then it will include as a portion of it the m-th best path from $(0, 0)$ to (i, j), where $m <= n$.

This states that a path which is not among the local top m can never be part of the global n-th best path. Therefore, determining the N best paths at each point in the DP algorithm will be sufficient to obtain the N best globally optimal paths. Although an implementation of this algorithm did produce the N best optimal phoneme alignments, virtually all of the alignments produced identical template sequences and many of the alignments even had the same score. The differences between the phoneme alignments were very small and usually consisted of just a few alternative phoneme substitutions. In one case the optimal alignment contained the substitution (t.d), whereas in the second best alignment this was replaced by the substitution (t.p) Apart from this minor difference all other substitutions were identical. This is not particularly useful for recognition purposes since we are usually more interested in the sequences of word templates rather than the low level details of the underlying sequence of phoneme substitutions. Rather than generating N different alignments of the same template sequence, it

would be much more useful if we were able to obtain the N-best different template sequences.

The N-best algorithm [Schwartz & Chow 87] achieves this goal by taking the history of the paths into account when they are being propagated by the DP algorithm. The history of a path is a label which uniquely identifies the sequence of templates that lead up to the onset of the path. Backtracking along paths that have the same histories will result in identical sequences of templates so only a single instance of any path with a given history needs to be recorded. Whenever two or more paths meet at the same point in the DP matrix their histories are compared and, if they are the same, only the best scoring path is kept and the other path is discarded. If the histories differ then both paths are kept, subject to the limit N on the maximum number of paths, where N is the number of template sequences that are required.

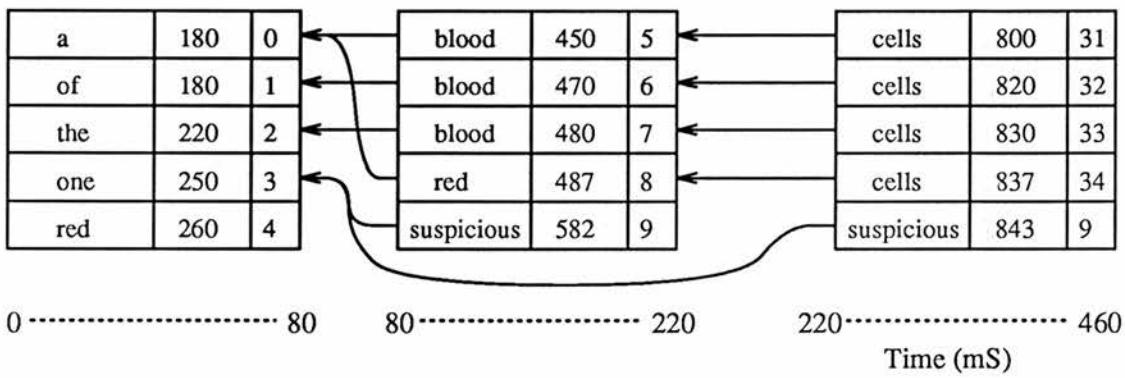


Figure 4–1: Backtracking Information Stored by the N-best Algorithm

Once all paths have been processed in the current time interval the N best scoring templates are determined as before by comparing path scores from the last frame of each template. Backtracking information is stored as depicted in figure 4–1. In this figure the five lowest scoring templates have been stored at each interval. Each entry contains the template name, path score, and a unique sequence number as well as a backpointer which indicates the starting time and rank of the preceding template. The sequence number uniquely identifies each par-

tial sequence of templates so that different partial sequences such as 'the blood' and 'a red' have different numbers whilst the two occurrences of the partial sequence 'one suspicious' have the same sequence number. The sequence numbers are used to specify the history of all paths which originate from the template when new word initial paths are propagated by the DP algorithm. It should be clear that an extended DP algorithm will calculate the optimal extension of each path subject to the constraint that each of the N-best paths at a given point in the matrix has a unique history. Therefore, it follows that, once the input lattice has been consumed, all of the $n = 1 \ldots N$ paths will produce a different template sequence on backtracking and that the single phonemic alignment of each template sequence will be optimal.

4.2.1 Calculating and Accessing Sequence Histories

When we come to assign a sequence number to each partial sequence of templates we need to be able to quickly determine whether this particular sequence has been observed before so that we can ensure that a unique label is assigned to each sequence. This could be achieved by storing a tree of all partial word sequences seen so far [Steinbiss 89] and then comparing the partial sequence against successively deeper nodes in the tree. If this is the first occurrence of the sequence a complete match will not be possible so the template sequence would be added as a new branch to the tree along with a unique sequence number. However, if the tree already contains the partial template sequence the number that has been assigned to this sequence will be retrieved from the last node to have been successfully matched and returned.

The same results can be obtained much more efficiently if we observe that two word sequences will be identical if, and only if, they end with the same template and the alignments within each of the templates have the same history. I.e. two sequences will be identical if they have the same final template and the same sequence of preceding templates. Hence equality can determined by a single comparison of the final template and its history rather than searching through the tree

and matching each of templates against the corresponding nodes. For example, in fig. 4–1 we would find that the two occurrences of the word sequence 'one suspicious' are identical since they both have the same final template 'suspicious' and each of the alignments within these two templates have the same history (3) which identifies their common preceding sub-sequence 'one'.

In order to do this efficiently we collect the previously observed word sequences into groups according to their length, where length is defined as the number of templates in the partial word sequence. Each sequence of words in a group is identified by a pointer to the final template and an integer which specifies the history of its alignment. Then, when we need to determine whether a particular sequence has been previously observed we first determine its length (which is simply one plus the length of its preceding sub-sequence) to find the appropriate group of word sequences and then test all of the words in this group for equality with the last word in the sequence by comparing spellings and histories. If the sequence has been observed previously a successful match will be found and its sequence number will be returned otherwise the sequence will be added to the list and assigned a unique number. The storage requirements are the same as those of the tree structure but access is much faster since only a single set of templates have to be compared rather than the numerous potential nodes along a branch in the tree.

4.2.2 Path Regeneration

The standard N-best algorithm as outlined above will calculate and store the N-best partial paths at each point and therefore the increase in computational and storage requirements will be proportional to N. This is a serious drawback since even in the basic DP algorithms the computational and storage requirements can be prohibitively large. The sub-optimal variants which have been proposed [Steinbiss 89], [Marino & Monte 90] calculate fewer paths (usually just one) within templates and then only determine the N-best paths at template boundaries. This significantly reduces the requirements but although the algorithm will still obtain

N reasonable alignments they are unlikely to be the genuine N optimal alignments. This limitation occurs because the sub-optimal variants construct their choice of the N best scoring templates straightforwardly by comparing the scores of the single alignment of each template in the lexicon. This means that a template can never occur more than once in the final list of the N best templates. Therefore, when there should be multiple occurrences of a template (such as 'cells' in fig. 4–1) the choice of the N best templates will be incorrect and the genuinely optimal alignments will be lost.

We have implemented the standard N-best algorithm which has been used to determine the 10 best optimal template alignments by calculating the 10 best partial paths at each point in the DP matrix. At the end of each time interval this algorithm determines and stores backtracking data about the N lowest scoring templates. An examination of these word lists showed that multiple instances of the same template were often present even though each of sequences of which they were a part had to be different. Furthermore the starting times, and of course the finishing times, of the optimal path within these templates were usually identical. Since each path must have a different history the paths must have originated from different templates in the same word list stored at some preceding time interval (e.g. 'blood' and 'cells' in fig. 4–1).

The N-best algorithm will calculate the N optimal extensions of each path, each of which will have different histories. When a number of paths having different histories begin at the same time the N-best algorithm will calculate the optimal extension of each path. If these paths also reach the final frame of the template at the same time the sequence of phoneme substitutions along each of the optimal paths will be identical since each path will have consumed identical template and input phonemes. Because the same sequence of phoneme substitutions have occurred along each of the paths the template alignment score (i.e. final path score – initial path score) will also be identical. This result can be seen in figure 4–1 where the template alignment score for the alignment of 'cells', starting at 220mS and finishing at 460mS, is always 350 (e.g. 800 – 450 or 830 – 480) irrespective of the initial path score.

Since the sequence of phoneme substitutions along each template alignment must be identical if each of them are to be optimal it should be possible in principle to only calculate one of the template alignments. The scores of the missing alignments can then be regenerated from the score of the single template alignment and the scores of the remaining templates in the preceding word list. We would therefore calculate a single alignment of the template 'cells' whose template alignment score would be 350. The scores of the missing alignments are then equal to the score of the single alignment plus the scores of the other templates in the preceding word list giving a score of 800 (350 + 450) for the second occurrence of 'cells' and 830 (350 + 480) for the third.

By regenerating the missing alignment scores we will be able to obtain the genuine N best template sequences by only calculating single alignment in the cases where two or more of the N best template alignments started and finished at the same time. The regeneration algorithm is described below. In order to illustrate the algorithm we will explain how it would be used to regenerate the list of optimal templates in the last word list of fig. 4–1 from the intermediate word list shown in fig. 4–2.

1. Use the standard N-best algorithm to calculate a small number N (e.g. 2) of paths at each point within each template.

2. Construct a temporary word list which contains the M (e.g. 10) lowest scoring paths from the N paths within each template.

3. For each template in this list, find the list of preceding words from which each of the alignments originated.

4. Regenerate the path score that would have resulted if the actual alignment had been extended from each of the words in this list.

5. Reconstruct the word list containing the M-Best templates using the contents of the temporary word list and the regenerated path scores from step 4.

The temporary word list in figure 4–2 contains the $M = 5$ lowest scoring paths chosen from the $N = 2$ optimal paths in each template [1]. Each template in this list is examined in turn and the preceding list of words is located from the backtracking pointer. For example, the list of templates which can precede the first occurrence of 'cells' is identified by the backpointer and the template alignment score of this word can be calculated as being 1350 (1800 - 450). The path scores can now be regenerated for each of the paths that would have been obtained if the alignment had started from the remaining templates in this word list. This is achieved by adding the original template alignment score of 'cells' to each of the path scores in the preceding list. This gives alternative path scores of 1820, 1830 1837 and 1932 using each of the remaining templates in this list. Similar scores can be calculated using the second occurrence of 'cells' and the other templates in the temporary word list. The M lowest scoring paths are then rechosen by comparing the scores of the original paths in the temporary word list with the scores of the regenerated paths. This will result in the creation of the final word list shown in figure 4–1 which contains the five genuinely lowest scoring paths.

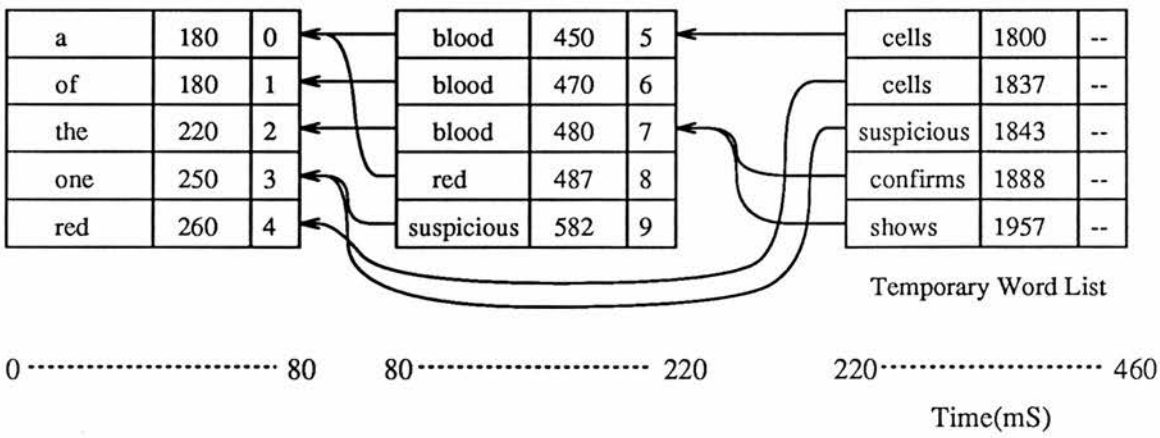


Figure 4–2: Backtracking Information Stored by the Regeneration Algorithm

[1] This is also the same word list that would have been generated by the sub-optimal N-best variants and then used in subsequent stages of the DP alignment process

We have collected statistics using the standard N-best algorithm (with $N = 10$) for lexical access on the previously mentioned sentences from the cytology domain. At the end of each time interval the program examined the intermediate word lists and looked for multiple occurrences of a template. Every template that occurred more than once was then examined in more detail and the number of alignments that had different starting times was counted. This total indicates the minimum number of optimal paths that would have to be calculated in order to ensure that the genuine N-best paths could be regenerated. For example, in figure 4–1 there are four instances of the template 'cells' but the four alignments only have two different starting time (80mS and 220mS). Only two optimal alignments need to have been calculated since the three alignments that started at 220mS could have been regenerated from one of the alignments that also started at that time.

We found that at least 80.5% of the genuinely optimal ten best template sequences would have been obtained without error if we had calculated just one path within each template and regenerated the rest. This increased to 99.6% of the ten best sequences when the two best paths are calculated within each template and 100% for 3 best paths per template. An implementation of the path regeneration algorithm described previously has confirmed these empirical results.

This technique is only slightly more complicated than the sub-optimal variants that have been previously proposed. These would immediately use the temporary word list (fig. 4–2) when extending paths in the next time interval and as a result would lose many of the optimal paths that could have been regenerated. It is also interesting to note that other definitions of the template sequence number and path histories may lead to useful results. If we were to define the sequence number to be equal to the number of templates in the sequence, the N-best algorithm would produce the N optimal alignments subject to the constraint that each of them contains a different number of templates. This may be more useful than the usual level building approach [Myers & Rabiner 81] since we would not be forced to produce a number of improbable alignments that contain very few templates (i.e. 1 upwards) or as many as the maximum number of levels in the level building algorithm.

4.3 Beam Width Pruning

We have investigated the extent to which beam width pruning could be used to reduce the search space. The standard N-best algorithm described previously has been implemented and applied to our lexical access domain and evaluated using a jack-knifing procedure on four different partitions of 20 evaluation sentences from the cytology domain to increase the reliability of the statistics. The procedure described below was used to gather the statistics which allowed empirical values for the beam width to be calculated for a given trade-off between search space reduction and preservation of the N-best optimal paths.

The standard N-best algorithm was used with values if N ranging from one to ten. At the end of each time interval every template was automatically examined by the program and the best scoring path from amongst all paths in the current time interval (the number of paths being equal to the number of template phonemes * N of the N-best algorithm) was determined. The differences in accumulated distance scores between this locally optimal path and every other path in the current time interval was then calculated. These score differences were then used to update records, stored on each path, which contained the maximum score deviation of the path from the locally optimal path. A count of the total number of paths falling within a range of beam widths from the locally optimal path was also updated and recorded.

The results are summarised in figures 4–3 to 4–5 for values of N of one, two, five, and ten. The accumulated distances scores calculated by the DP lexical access algorithm are equal to the sum of the scaled negative log probabilities ($-128 * \log_2$ probability) of observing each of phoneme substitutions from which the paths are comprised. Therefore, differences in accumulated distance scores ought to reflect the relative probabilities of the two paths. For example, a beam width of 1280 would imply that all paths having scores within the beam would not be more than 2^{10} or 1024 times less probable than the locally optimal path.

Figure 4–3 shows the percentage of the best N paths whose maximum score deviation from the locally optimal path was less than the specified beam width throughout their length. From this figure we can see that 90% of the best paths and 80% of the best 10 paths had a maximum score deviation of 1200 from the locally optimal path at some point during their alignments.

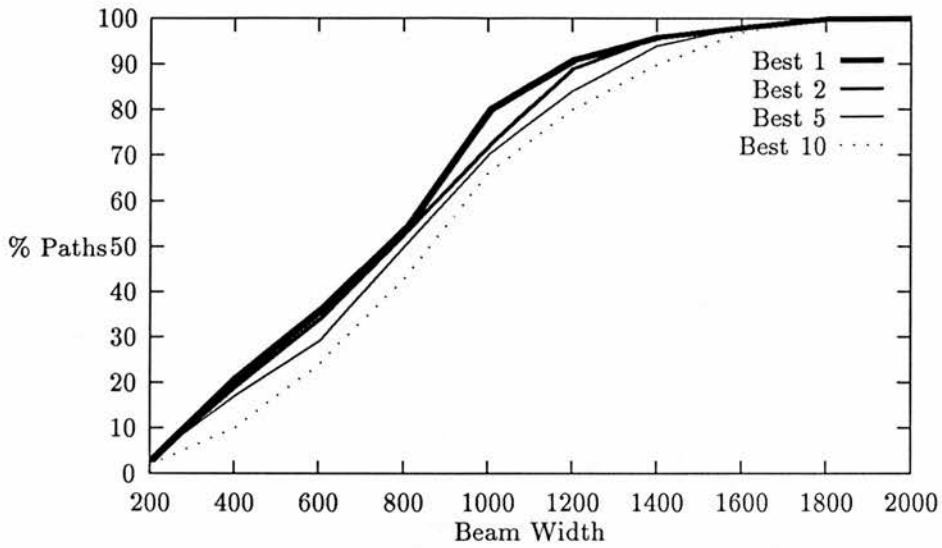


Figure 4–3: Percentage of the N best paths whose maximum score deviation from the locally optimal path is less than the beam width

The graphs show that all of the 10 best paths could have been found within a beam width of 2000 from the locally optimal path and that very few paths, including the globally optimal path, remain close to the locally optimal path throughout their length (only 20% of the globally optimal paths remain within a beam width of 200). The fact that the graphs are closely spaced and have similar shapes, particularly the linear onset, would seem to imply that the paths are evenly distributed throughout the beam width. It does not appear to be the case that the globally optimal path is consistently closer to the locally optimal path than the 2nd, 5th or 10th best paths which would seem to limit the scope for beam-width pruning. However, if a beam width is to be used to prune the search space the graph can be used to predict the probability of obtaining the correct set of N-best paths for a given beam width.

The percentage of all paths which remained within a given beam width of the

locally optimal path are shown in figure 4–4, this includes all of the paths which make up the search space including the best N paths which were shown in the previous figure. From this graph we can see that approximately 30% of the paths in the N-best algorithm lie within a beam width of 1200 from the locally optimal path when $N = 10$ whereas 37% lie within this beam width when $N = 1$. By combining these results with those in figure 4–3 we can see that setting a beam width of 1200 would reduce the search space of the best-10 algorithm by 70% whilst retaining 80% of the genuine ten best paths. A comparison of the graphs in figures 4–3 and 4–4 shows that there is an optimal value for the beam width which lies between 1200 and 1400. Reducing the beam width to 1400 has a comparatively small effect on the probability of obtaining the genuinely optimal paths (from fig. 4–3) but results in a significant reduction in search space (from fig 4–4). However, reducing the beam width below 1200 begins to cause a much larger reduction in the probability of obtaining the genuinely optimal paths but without a corresponding increase in the reduction of the search space.

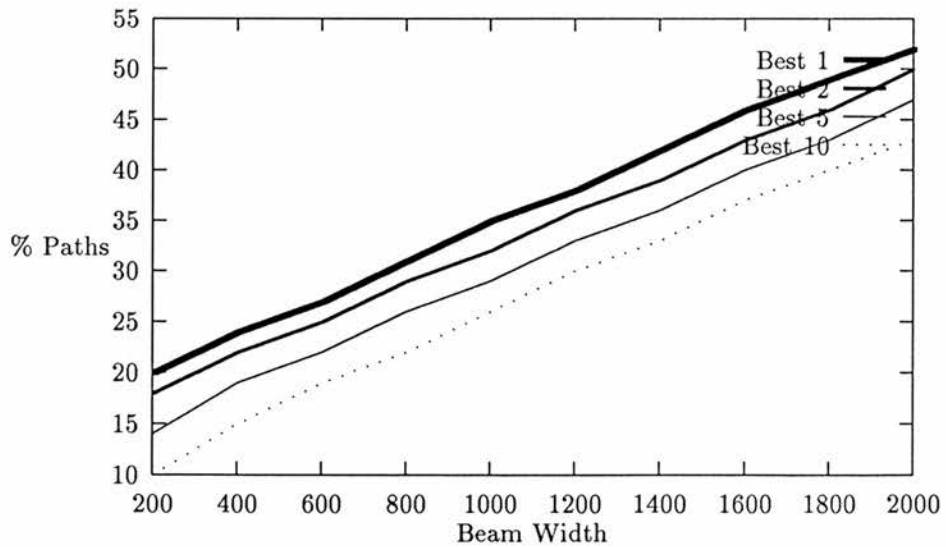


Figure 4–4: Percentage of all paths whose maximum score deviation from the locally optimal path is less than the beam width

Finally, figure 4–5 shows how often we would expect the n-th best path (where $n = 1 \ldots N$) to be present in the 10 alignments produced by the N-best algorithm if we were to prune with the beam widths specified along the x axis. This graph

shows that if we used the N-best algorithm to determine the 10 best paths and prune intermediate paths using a beam width of 1400 that 90% of the genuinely optimal and second best template sequences would be present. We would also expect to find around 85% of the genuine 5th best sequences and 80% of the genuine 10th best sequences. It should be noted however that even with pruning a N-best algorithm will always return N alignments, although some of these alignments, as well as their ranks, may be different to those that would have been obtained without pruning.

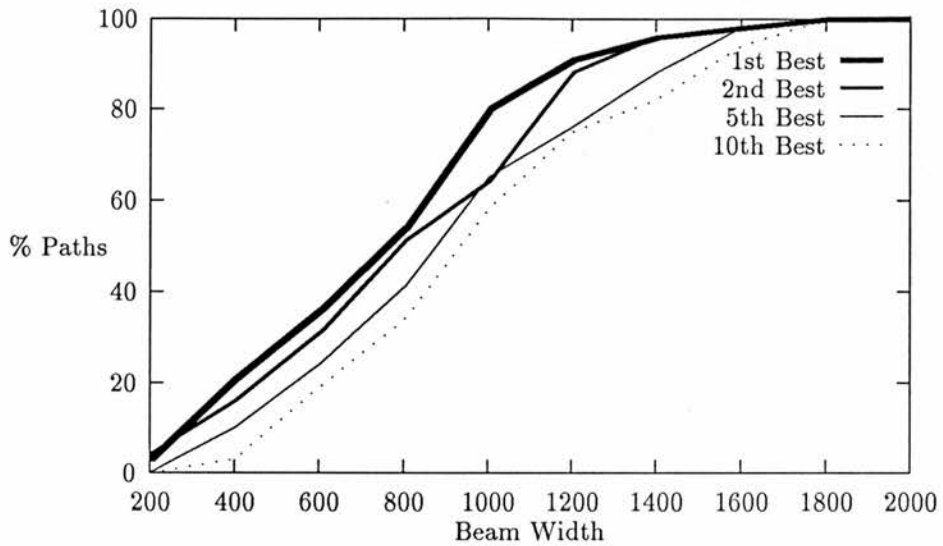


Figure 4–5: Probability of observing the genuine n-th best path when beam width pruning the N-best algorithm (N = 10)

The shapes of the plots in these three graphs are perhaps surprisingly similar and reasons for this became apparent when we examined the actual words which make up the best N template sequences. We found that most of the N-best sequences were similar, differing in perhaps one or two words only. The differences in accumulated distance scores between the optimal and nth ($n = 1 \ldots N$) best sequences were typically much less than the maximum deviation of any sequence from the local best path. This would seem to indicate that there is still a large amount of uncertainty in the choice and relative orderings of the alignments so that the 10th best alignment is not obviously worse than the optimal alignment.

Chapter 5

Applying Syntactic Constraints

5.1 Introduction

The use of grammatical knowledge to constrain the search space has been shown to be an extremely effective technique for improving the performance of speech recognition systems and it is often only by including such knowledge that satisfactory results are obtained [Lee 88]. The most successful large vocabulary recognition systems employ a tightly constrained finite-state grammar specialised for a particular domain. In these systems the lexicon is partitioned into a number of syntactic/semantic classes each of which contain relatively few word templates. The grammar constrains the relative ordering of the classes and usually only a few classes are syntactically valid at each stage of the recognition process. In this way the large vocabulary speech recognition problem is sub-divided into a series of small vocabulary speech recognition problems. Since the number of templates that have to be considered at each stage is much smaller recognition is faster and more accurate.

However, the danger with a highly restrictive grammar is that it is possible to end up with a system that is too rigid and which only 'hears' what it expects to hear, regardless of the acoustic input. When a disproportionate amount of emphasis is placed upon syntactic constraints the performance of the acoustic front end becomes less important and any improvements in its performance may have little or no effect upon the final recognition rate.

In this thesis we are principally concerned with the extraction and run-time application of phonological knowledge. Therefore we have not devoted a great deal of

effort to produce a highly specialised grammar and parser with impressive predictive capabilities since this would inevitably limit the scope for potential improvements in AFE performance which may arise due to the application of phonological knowledge. Instead, we have chosen to use a flexible chart parser which allows the interaction between the application of syntactic and phonological knowledge to be investigated. Of course our recognition scores could be significantly increased by tailoring the grammar to our medical domain of cytology utterances but this would seem to be counter productive for our present purposes.

5.2 Active Chart Parsing

The chart parsing module is an extension of MChart [Thompson 83]. At the heart of the parser is a data structure known as the 'chart'. This is a single data structure that is used to record lexical word hypotheses, partially hypothesised constituents, well formed constituents and complete parses. The chart (e.g. fig. 5–2) is composed of a sequence of vertices which are connected by active, inactive and lexical edges.

Inactive edges represent complete grammatical constituents such as adj(blood) and np(adj(blood)n(cells)) whereas active edges represent the grammar rules such as s $\Rightarrow$ np()vp() and np $\Rightarrow$ adj(blood)n(). Each active edge records information about the constituents found so far and the outstanding constituents that are required to satisfy the grammar rule. Each of the active edges also has an extension requirement which records the first outstanding constituent. This is the category which is immediately required by the edge in order to extend the (partial) parse described by the grammar rule.

The chart parsing process proceeds by constructing and adding new active and inactive edges based upon the prior contents of the chart. Lexical edges are added as word hypotheses are received from the lexical access component which in turn cause new active and inactive edges to be added to the chart. A fundamental

principle of this process is that existing edges are never modified or deleted once they have been added to the chart.

The parsing policy outlined below implements a top down parser by repeatedly expanding the non-terminal categories of the initial top level sentence edge and subsequent active edges. The implementation of this parsing policy causes additional active edges to be added to the chart which could satisfy the extension requirements of the original active edge.

```
Whenever an active edge is built with right vertex V
   if    its extension requirement E is expandable
         by the grammar rules
   then  add an empty active edge at V for each grammar rule
         which could produce E.
```

Figure 5–1: Top Down Parsing Policy

Whenever an active and inactive edge meet at a vertex for the first time the extension requirement of the active edge is compared with the grammatical category of the inactive edge. If the category satisfies the extension requirement a new edge is created. An inactive edge will be created if no outstanding constituents remain otherwise a new active edge will be created which contains updated information about the completed and outstanding constituents. The addition of one edge may in turn provoke the addition of many more edges if for example, a new inactive edge meets the extension requirements of several existing active edges, or there are several grammatical rules which could be used to expand the extension requirements of a new active edge.

The chart parsing process can be illustrated using fig. 5–2. The top level active edge s $\Rightarrow$ np() vp() will be the first edge to be added to the chart. The top down parsing policy will then expand the extension requirement of this edge by adding the active edge np $\Rightarrow$ adj() n() which would satisfy the extension requirement of the original edge if completed. The terminal extension requirement of this edge can

not be expanded by the top-down parsing policy so the process of edge expansion now terminates.

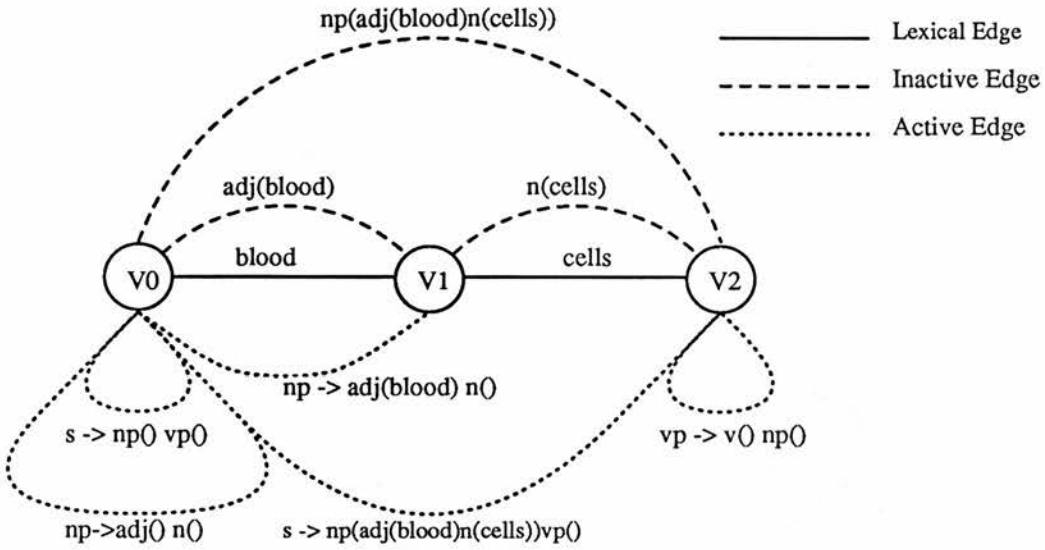


Figure 5–2: An example of a small chart fragment

The addition of the inactive edge adj(blood) will satisfy the extension requirement adj() of th active edge np ⇒ adj() n(). This causes a new active edge np ⇒ adj(blood) n() to be created which has the category 'n' as its extension requirement. This active edge is completed when the addition of the inactive edge n(cells) causes the last extension requirement to be satisfied. The inactive edge np(adj(blood)n(cells)) is then added to the chart to represent the completed constituent. This new inactive edge now satisfies the extension requirement of the original top level active edge and a new active edge is created which consumes this constituent. The updated extension requirement vp() of this edge is expanded by the top down parsing policy as before. Parsing continues in this manner with the parser currently waiting for an inactive v() edge.

5.2.1 MChart - A Modular Chart Parsing Skeleton

Although MChart [Thompson 83] is not a parser in itself it does provide an elegant and flexible framework for the construction of chart parsers which allows a number of different parsing strategies to be easily implemented. MChart uses an agenda

along with a scheduling mechanism to separate the process of edge creation from their addition into the chart and thereby allow edges to be added to the chart in a different order to that in which they were created.

Further flexibility is provided by the use of signals which are raised at numerous points throughout the parsing process such as whenever an active or inactive edge is about to be added to the chart. Whenever a signal is raised by the parser a predefined response is made which may involve calling a default function, a user specified function, or possibly ignoring the signal completely. The top-down parser is implemented by attaching a function to the active edge signal which, when raised, causes additional active edges to be created that could satisfy the extension requirement of the original active edge.

The agenda consists of a list of queues which are stored in order of decreasing priority. Each of the queues has its own set of functions which are used to schedule and remove entries. Default functions are provided that implement LIFO (Last In First Out) and FIFO (First In First Out) queuing strategies but these can be replaced by more complex queuing strategies if desired. The choice of queuing strategy determines the search strategy of the parser. For example a LIFO active edge queue will add the most recent active edges to the chart and will produce a depth-first search whereas a FIFO queue will add the oldest edges to the chart and this will produce a breadth-first search. In the following section we will describe how a more complex queuing strategy is used to perform dynamic programming and thereby obtain the optimal parse(s).

Each of the entries in a queue is in fact a delayed function call which consists of a collection of pointers to the function to be called and its parameters. When an entry is removed from a queue the pointers are retrieved and the parameters are passed to the function which is now executed. The agenda contains queues for functions which add active edges, inactive edges, and lexical edges as well as a low priority master queue for functions which handle housekeeping tasks such as initialising the chart and displaying the parses.

The chart parser proceeds by executing the first entry on the highest priority non-empty queue. Execution of this function will typically cause an edge to be

added to the chart which will in turn cause the addition of further edges to be scheduled onto the appropriate queues. This cycle continues until all of the words have been added to the chart by which time all possible parses will have been extended. Finally, when no more entries remain on the higher priority active, inactive, and lexical edge queues, an entry from the low priority master-queue is removed and executed. This entry causes each of the syntactically valid parses to be displayed.

5.3 Dynamic Programming

The primary function of the chart parser is to constrain the DP lexical access algorithm so that only syntactically valid template sequences are generated by the DP alignment process. At the end of each time interval the lexical access algorithm will propose a number of word hypotheses each of which have a score that reflects the likelihood of the word hypothesis given the acoustic evidence. The chart parser uses the scored hypotheses to extend existing partial parses and determine the set of terminal categories that will be syntactically valid during the following time interval. These categories are then sent back to the lexical access component which uses the information to ensure new paths are only propagated between syntactically valid categories.

It is necessary to incorporate a dynamic programming algorithm into the chart parser so that the best partial alignments are extended in the correct manner and the optimality of the DP alignment process is preserved. Consider for example the hypothetical chart fragment in fig. 5–3 which for simplicity only shows lexical and active edges. Scores are assigned to active and inactive edges which are derived from the word scores provided by the lexical access component. The score of each lexical edge is equal to the score of the alignment between the word template and the portion of the phoneme lattice which it spans. This alignment score is calculated by the DP algorithm (see Ch. 3) and is equal to the scaled negative logarithm of the probability of the alignment given the acoustic evidence.

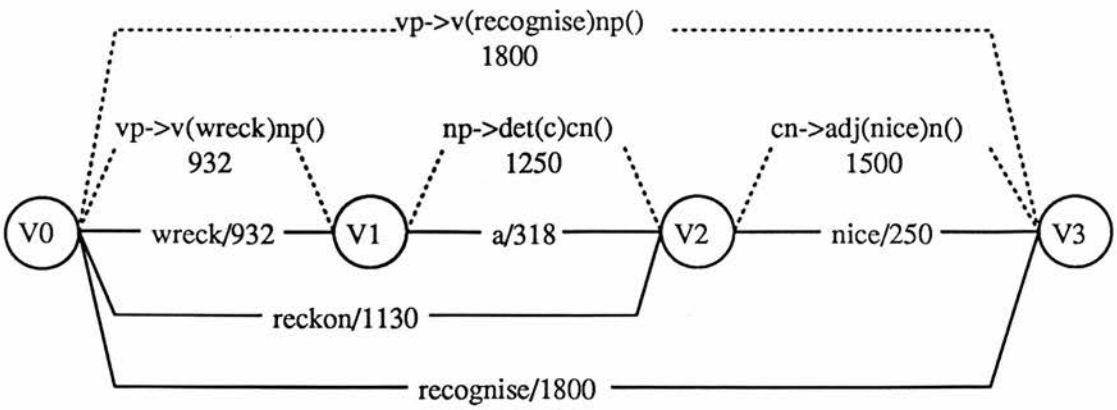


Figure 5–3: Chart fragment showing partial path scores

When active and inactive edges combine to produce a new active edge the score of the new edge is simply calculated as the sum of the scores of the active and inactive edges. The probability of the new edge is therefore equal to the product of the probabilities of the individual active and inactive edges. The final score of the top level sentence edge will be equal to the sum of the alignment scores (i.e. the product of the alignment probabilities) of the individual words which make up the sentence. Provision has also been made for an additional penalty to be incorporated whenever active and inactive edges combine. This penalty would also be equal to the scaled negative logarithm of the probability of combining the edges could be used in the future to implement a parser which uses a stochastic or N-gram grammar.

The initial score of a new active edge should be equal to the lowest score of the active edges from which the new edge could be generated by use of the top down parsing policy. This ensures that the score of an active edge represents the total alignment score from vertex V0 regardless of the vertex from which the active edge actually originated. Simply using an initial score of zero would unfairly penalise active edges that spanned larger portions of the chart (and therefore have a higher score) and would also ignore the score of those edges which precede the active edge.

Inactive edges however need to have a score which reflects only the portion of

the input which they span otherwise the alignment scores would not be calculated properly when active and inactive edges were combined. Therefore each active edge also records the initial score which was assigned to it when it was created. This score is then subtracted from the final score of the active edge when it is complete to give the correct score for the inactive edge which is added to the chart.

Each active and inactive edge in the chart also records the most recent terminal category (MRTC) to have been consumed by the edge, or in the case of a new active edge, consumed by its parent edge. The MRTC along with the terminal extension requirement of the edge specify how paths should be propagated by the lexical access component across word boundaries. For example, the MRTC of the active edge s $\Rightarrow$ np(adj(blood)n(cells))vp() is simply 'n'. The non-terminal extension requirement vp of this edge will be expanded by the top-down parsing policy producing a new active edge vp $\Rightarrow$ v()np() whose MRTC will also be 'n'. The extension requirement of this edge is 'v' and it would therefore be syntactically valid for the lexical access component to propagate alignments between templates of categories 'n' and 'v'.

Whenever an active and inactive edge combine in the chart the MRTC of the resulting edge is equal to the MRTC of the inactive (i.e. rightmost) edge. The initial MRTC of the top level sentence edge and the new active edges which are derived from it by expanding its extension requirement are equal to the pseudo category 'NULL'. An edge whose MRTC is 'NULL' has not consumed any terminal categories nor has its parent and can therefore only occur at the start of a parse. The extension requirements of these edges identify categories containing templates that can be aligned at the start of an utterance.

In figure 5–3 there are two partial paths to vertex V3 which have scores of 1800 (recognise) and 1500 (wreck a nice). Both of the active edges which terminate at V3 have the same extension requirement 'n()'. It would be syntactically valid for the lexical access component to extend either of the DP alignments of 'adj(nice)' or 'v(recognise)' into templates of category 'n'. In order to preserve the optimality of the DP alignment process new alignments of templates in category 'n' should

be propagated from the optimal (lowest scoring) alignment in the preceding time interval (i.e. from adj(nice) rather than v(recognise)). The optimal path to each vertex, for a given extension requirement, can be calculated efficiently using dynamic programming techniques which incrementally construct the optimal path to each vertex upon the basis of the optimal paths to each of the preceding vertices.

5.3.1 The Algorithm

A dynamic programming algorithm can be implemented by replacing the simple FIFO active edge queue with a two level structure [Thompson 89], [Thompson 90] as shown in figure 5–4. The first level queue contains entries for active edges with non-terminal extension requirements (np, vp etc) whilst the second level queue contains entries for active edges with terminal extension requirements (n, v, det etc). We have further organised the entries in the second into groups containing edges which have the same extension requirements. Edges in each of these groups are stored in order of increasing score.

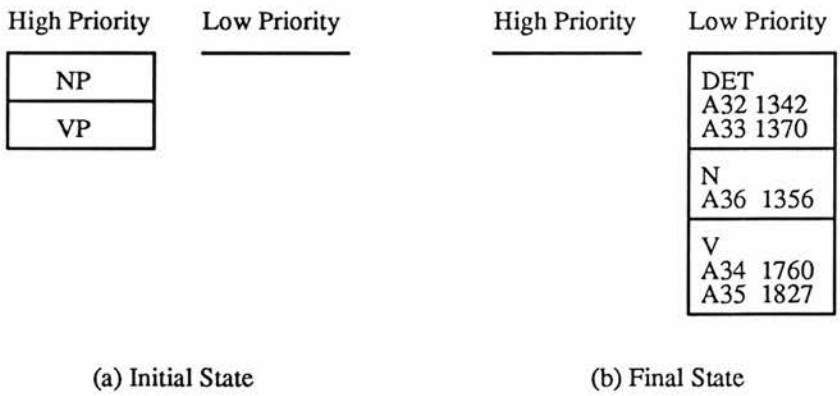


Figure 5–4: The initial and final states of the two level active edge queue used to implement the dynamic programming algorithm

The second level queue has a low priority so that edges will be added to the chart from the first level and other higher priority queues before entries from the second level queue are processed. Initially the second level queue will be empty (fig. 5–4(a)) but as edges are taken from the high priority queue new entries

will be scheduled on the low priority queue due to the expansion of the non-terminal extension requirements by the top down parsing policy. Eventually the high priority queue will become empty and the only outstanding edges will be those residing on the second level queue (fig. 5–4(b)). All of the remaining edges on the second level queue have extension requirements that are terminal categories.

When the only outstanding edges are those residing on the second level queue we can be sure that the contents of the second level queue are stable. The addition of edges from this queue will not cause further edges to be scheduled since the extension requirements of these edges can only be satisfied by the word hypotheses which will be produced by the lexical access component at some future time. By delaying the addition of edges from the second level queue we also ensure that the extension requirements of the outstanding edges represent the *complete* set of syntactically valid terminal categories and that *all* edges that have these extension requirements are present in the second level queue.

The lowest scoring edge in each group identifies the optimal path which should be extended when creating new alignments of templates in the category corresponding to the extension requirement of the group. Information about the MRTC and extension requirement of the lowest scoring edge is passed to the lexical access module where it is used to ensure that paths are propagated between syntactically valid categories and with the optimal path scores. The remaining entries in each group are discarded.

The DP algorithm used by the chart parser operates time synchronously from left to right in step with DP lexical access algorithm. As each segment in the input lattice is consumed by the DP lexical access algorithm a number of scored word hypotheses are sent to the chart parser which causes lexical edges to be added to the chart. The vertices which these edges span are determined by the starting and finishing times of the alignment within each word template. As the DP lexical access algorithm also operates time synchronously from left to right each template alignment will have the same finishing time and the corresponding lexical edges will end at the same vertex (say V_e) in the chart. The starting time

of each alignment and hence the initial vertex (V_s) of each lexical edge will usually be different.

The addition of lexical edges to the chart which span vertices V_s to V_e will allow active edges terminating at V_s to be extended provided that the extension requirements of the edge match the syntactic category of the word. As described previously this will cause new active and inactive edges to scheduled for addition to the chart in the first and second level queues which may in turn cause additional edges to be scheduled. Each of these edges will also terminate at vertex V_e since they will have been produced by directly or indirectly consuming the new lexical edges which terminate at V_e. This process will stop when the only outstanding edges are those remaining on the second level active edge queue. All of the edges in the second level queue will therefore also terminate at the same vertex V_e. Information about the lowest scoring edge in each group is passed to the lexical access component where it is used to control the propagation of alignments between word templates in the following time interval. The whole process is then repeated when the lexical access component once again provides word hypotheses at the end of the following time interval which now terminate at $V_e + 1$.

5.3.2 Allowing for Phonological Knowledge

When we come to apply phonological knowledge we must take even more care when combining active and inactive edges in the chart. In this situation the scores that are assigned to particular word hypotheses by the DP alignment process may be conditional upon the presence of particular sequences of phoneme substitutions in the preceding words. For example, the DP score of the alignment (s.-)(p.p)(ii.ii)(ch.ch) of the template 'speech' might be much lower if the preceding template 'fast' ends with the substitutions (s.s)(t.-) so that the MSub [(s.s)(#.-)(s.-)] was applicable. The lexical access module does not know how the word hypotheseses which it sends to the parser will be combined in the chart so the parser itself must ensure that the word hypotheses are combined in the correct manner and with appropriate scores. However, it does not seem reasonable to

require that the parser be concerned with low level details such as the particular pronunciation of each word and the set of phonological transformations that may span word boundaries along with their effects on the subsequent alignment scores.

A pair of tags (Ltag Rtag) are added to each active, inactive and lexical edge which provide additional constraints on the combination of edges in the chart. Each tag is simply an integer which identifies a particular sequence of phoneme substitutions. Non-zero tags uniquely identify those sequences of substitutions which precede internal word boundary deletions (#.-) in MSubs. For example, the tag for the sequence of substitutions (s.s)(t.-) (from [(s.s)(t.-)(#.-)(s.-)]) might be assigned a value of 4 whereas the tag for the sequence (s.s) (from [(s.s)(#.-)(s.-)]) might be 5 and so on. When the value of the tag is 0 the sequence of substitutions is undefined and may or may not correspond to substitution sequences which precede an MSub's internal word boundary deletion.

The Rtag of an edge indicates the phonological context that is *provided* by the template and identifies the sequence of substitutions at the end of the alignment (for example (s.s)(t.-) in an alignment of 'fast'). An Rtag whose value is non-zero indicates that the final substitutions in the alignment corresponds to the initial substitutions of a word boundary spanning MSub which could be applied during the propagation of an alignment in subsequent templates.

The Ltag of an edge indicates the phonological context that is *required* in the previous word and which were assumed to be present when the MSub was applied and the DP alignment score was calculated (the application of Msubs is described in chapter 7). The scores of alignments with non-zero Ltags are dependent upon the sequence of substitutions in the preceding alignment since these must provide the substitutions which match the initial substitutions of the MSub which was applied. By default, when no word boundary spanning MSubs are being applied, the Ltags and Rtags of all edges will be zero to indicate that there are no contextual constraints on how edges can be combined in the chart.

Whenever an active and inactive edge are about to be combined the Rtag the leftmost (active) and the Ltag of the rightmost (inactive) edge are compared. The edges are only be combined if the Rtag of the active edge is compatible with the

Ltag of the inactive edge. For the tags to be compatible they must have identical values which indicate that the required and provided contexts of the edges are identical or the Ltag must be zero.

An Ltag is zero when no word boundary spanning MSubs have been used in the alignment and thus the score is independent of the substitutions in the alignment of the preceding template. Since the score of the edge does not depend upon the substitutions in the alignment of the preceding template the Rtag of the preceding edge is irrelevant. However note that the reverse is not true and an active edge whose Rtag is zero is only compatible with an edge whose Ltag is also zero. The final substitutions in the alignment of an edge whose Rtag is zero are undefined. These edges should only be allowed to combine with edges whose Ltags are also zero and whose alignment scores do not depend upon a specific sequence of substitutions in the preceding alignment.

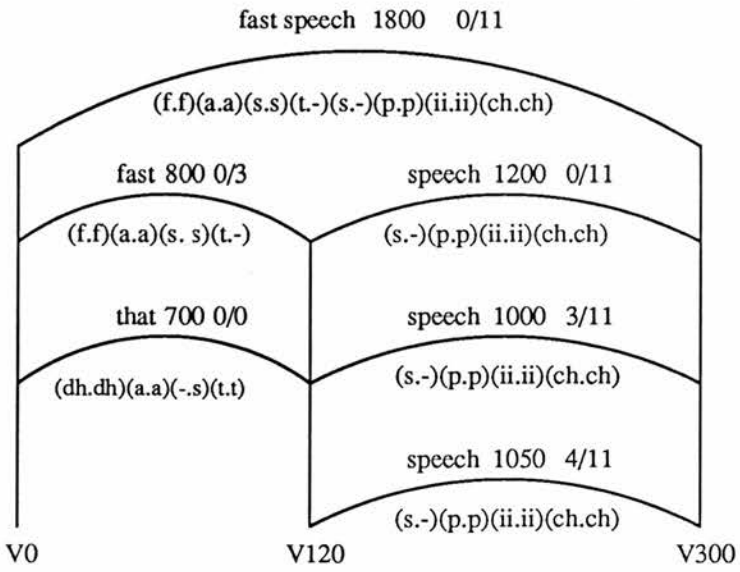


Figure 5–5: An example showing how the combination of edges is constrained by the presence of tags in order to enforce assumptions made during the application of cross word boundary.

If the active and inactive edges have compatible tags the parser constructs a new edge as described previously. The Ltag and Rtag of the new edge are taken from the tags of the parent edges and these provide the appropriate constraints for

future combinations. The new values for the Ltag and Rtag come from the Ltag (required context) of the active (leftmost) edge and the Rtag (provided context) of the inactive (rightmost) edge respectively. Hence, the Ltag of any edge will always specify the required context of a preceding edge and the Rtag will always specify the context that is provided for combinations with successive edges.

This process is illustrated in figure 5–5. The alignment of each template is also shown for clarity although in practice this information would not be available to the chart parser. In this example the lexical access module has hypothesised two different words in the first segment. The first of these is 'fast' which has a Rtag value of 3 which identifies the sequence (s.s)(t.-) which might be required by future words which use the MSub [(s.s)(t.-)(#.-)(s.-)]. In the next segment there are three different instances of the word 'speech' each of which have different alignment scores.

Although each of the alignments are identical the scores are different and depend upon the presence of particular substitutions in the alignments of the preceding templates. The alignment of the lowest scoring template was calculated under the assumption that the alignment in the preceding template ended with the sequence (s.s)(t.-) (Ltag 3). This template has a slightly higher score when the previous alignment was assumed to end with substitution (s.s) (Ltag 4) and has the highest score of all when no assumptions were made about the preceding template (Ltag 0). Each of these words have the same Rtag since each contains the same sequence of phoneme substitutions.

The optimal combination of these edges results in the creation of the edge for the phrase 'fast speech' which is constructed from the two edges whose Rtag and Ltag values equal 3. The syntactically valid phrase 'that speech' could score lower but this combination is ruled out because of incompatible tags. The Ltag and Rtag for the phrase 'fast speech' are taken from the two edges which were combined to form the edge and these would be used to restrict future combinations of this edge in the same way.

5.4 Communication and Synchronisation

The chart parser and lexical access modules are loosely coupled independent modules which communicate and synchronise their activities by passing messages. A number of advantages are gained by separating the parsing and lexical access modules. The lexical access algorithm is no longer concerned with the details of the parsing strategy or the grammar and likewise, the parser is no longer concerned with the details of the DP alignment process or the phonological knowledge used by the lexical access algorithm. This means that we can alter the grammar and parsing strategy without having to make any changes to the lexical access algorithm and vice-versa subject to the constraint that the message passing interface remains constant.

The two modules can also be executed on separate processors possibly in different computers. This would allow for example, a specialised high speed processor to be used for the relatively simple but computationally intensive lexical access algorithms and a cheaper general purpose processor to be used for the more complex, but computationally less intensive stages of syntactic and semantic processing. In the current system, both components run on separate machines containing general purpose processors. The lexical access component runs on a fast Sparcstation with large amounts of memory whilst the parser typically runs on a slower Sun 3/80. Both components communicate with one another over an ethernet connection.

5.4.1 Communication

The lexical access component sends messages to the chart parser at the end of each time interval which describe the best scoring template in each category. These messages are of the following form :-

```
(tstart tend lexeme score (Ltag Rtag))
```

Tstart and tend are the starting and finishing times of the optimal alignment of the template 'lexeme' and these identify the vertices in the chart which the lexical edge will span. The template alignment score (i.e. final alignment score – initial alignment score) is sent rather than the path score (i.e. accumulated distance score) so that the correct edge scores can be calculated. The final item (Ltag Rtag) identifies the phonemic context which the template requires and provides for future paths. These messages are received by the chart parser and are used to schedule the addition of lexical edges which span the appropriate vertices in the chart. A number of these messages are sent at the end of each time interval which describe the template(s) with the lowest alignment scores in each category. The chart parser in turn sends messages to the lexical access component as shown below.

```
(Lcat Rcat score)
```

The Lcat and Rcat fields in these messages tell the lexical access component that it is syntactically valid for new alignments of templates in Rcat to be propagated from alignments of templates in Lcat during the next time interval. The score field contains a penalty (scaled negative log probability) for concatenating templates in the two categories. Currently this is unused and is always equal to zero but it would have non-zero values if the parser were to implement a stochastic grammar or N-gram model of sentence syntax. There will be a number of messages of this form which may have the same Lcats or Rcats depending upon whether it is syntactically valid for a category to precede or be preceded by a number of other categories. Values for the Lcat and RCat come from the most recent terminal category (MRTC) and extension requirement of the lowest scoring edge in each group of the second level queue.

These messages inform the lexical access component of those categories containing templates between which it is syntactically valid to propagate paths. The messages are collected by the lexical access component at the end of each time interval and are used to construct for each Rcat, a list of valid categories from the Lcats. For example, the three messages (det n 0), (quan n 0) and (adj n 0)

would be combined to produce the list (n (det 0)(quan 0)(adj 0)) which says that new alignments of templates in the category 'n' can be propagated from any of the templates in categories 'det', 'quan' and 'n'. Similar lists are constructed for each of the Rcat categories and for each category c give the set of syntactically valid predecessor categories C_s. The syntactically valid predecessor categories are used by the DP algorithm when enforcing the between word syntax constraints as described in chapter 3.5.1.

5.4.2 Synchronisation

Synchronisation is achieved by the passing of a small number of control messages which can originate from either the lexical access module (init, eow, and eos) or from the chart parser (eox). The first message to be sent 'init' originates from the lexical access component and causes the chart and various statistics to be initialised in anticipation of a new parse. The chart parser schedules the addition of the top-level sentence edge which initiates the top-down parsing process. As described previously the chart parser determines the syntactically valid terminal categories and sends the appropriate messages to the lexical access module. The end of these messages is signaled by the message 'eox' (end of extensions). The lexical access module then extends the alignments between syntactically valid categories during the next time interval and when finished sends messages to the parser informing it of the best scoring template(s) in each category. When all of the categories have been processed the message 'eow' (end of words) is sent and the parser replies with the next set of extension messages. This cycle repeats until all of the segments of the input lattice have been consumed and then the lexical access component sends the final message 'eos' (end of sentence) which terminates the parsing process and causes the valid parses (if there are any) to be displayed. Parsing of subsequent sentences is once again initiated with the message 'init' and the process is repeated.

A simple transaction might go something like the following (providing that word boundary spanning MSubs are not being used) :-

DP	Chart
init	
	(NULL DET 0)
	eox
(0 10 the 100 (0 0))	
eow	
	(DET NOUN 0)
	eox
(0 20 the 200 (0 0))	
(10 20 cells 200 (0 0))	
eow	
	(DET NOUN 0)
	(NOUN VERB 0)
	eox
(20 30 died 300 (0 0))	
(20 30 are 200 (0 0))	
eos	
	(S (NP (DET the) (NOUN cells))(VP (VERB died)))
init	
	(NULL DET 0)
	eox

5.5 Conclusion

In this chapter we have described the implementation of a chart parser which operates time synchronously from left to right in step with the lexical access module. The parser uses a set of simple domain independent context free phrase structure grammar rules (app. A.2) and scored word hypotheses from the lexical access module. A dynamic programming algorithm was incorporated into the chart parser to ensure that the lowest scoring active and inactive edges are propagated to each vertex in the chart. This preserves the optimality of the DP alignment algorithm

which is used by the lexical access algorithm. Additional care has to be taken when combining edges if word boundary spanning MSubs are used since the edge scores are no longer context independent. This is achieved by adding Ltags and RTags which provide additional constraints on how edges are combined in the chart. The parser is able to determine all valid parses subject to the additional constraints but it is not concerned with the range or type of the phonological knowledge which gives rise to these constraints.

The chart parser and lexical access modules are loosely coupled processes which communicate with one another by passing messages. The chart parser does not care about the algorithm and internal data structures used by the lexical access module and likewise the lexical access module does not care about the parsing strategy or grammar which is implemented by the chart parser. This provides an extremely flexible framework since it is in principle possible to make major changes to either of these components without having to modify the other component so long as the message passing interface remains constant. It would therefore be possible to use alternative sets of grammar rules or even change the parsing algorithm without having to make any changes to the lexical access module. Likewise, it should also be possible to use alternative sets of phonological knowledge or make extensive changes to the DP alignment algorithm in the lexical access component without having to make changes to the chart parser or grammar.

It is also be possible to execute each of these modules on separate processors which can be chosen according to the needs of the individual algorithms. A specialised high speed processor could be used for the relatively simple but computationally intensive DP alignment algorithm whilst a general purpose processor could be used for the relatively complex but computationally inexpensive parsing algorithm. Although the chart parsing approach is computationally more intensive than other approaches much of this is hidden since it is possible to perform most of the computation in parallel with that of the lexical access component. This would become more significant if more complex grammars were used or semantic processing was introduced which would increase the relative computation time of the parser.

Chapter 6

Context-Sensitive Dynamic Programming

In this section we will outline the motivations behind the use of macro substitutions (MSubs) and their extraction. We also show how the context sensitive substitution probabilities affect the alignment probabilities resulting in a correlated sequence of phoneme substitutions (i.e. a macro-substitution). A macro-substitutions is defined as a correlated sequence of phoneme substitutions. Each macro-substitution has a single correction factor which reflects the degree of correlation amongst all of the substitutions in the sequence.

6.1 Macro-Substitutions

The accumulated distance score $D(S_1 \ldots S_n)$ of an optimal phoneme alignment is calculated by the DP algorithm as the sum of the scaled negative log a-priori substitution probabilities $p(S_i)$. The accumulated distance score is therefore directly related to the probability of the phoneme substitution sequence. Ignoring constant scaling factors we have :-

$$D(S_1 \ldots S_n) = \sum_{i=1 \ldots n} \log p(S_i) \tag{6.1}$$

$$p(S_1 \ldots S_n) = p(S_1) * p(S_2) * p(S_3) * \ldots * p(S_n) \tag{6.2}$$

In practice the product of the a-priori probabilities is likely to give a poor estimate of the true probability of the sequence $p(S_1, S_2 \ldots S_n)$. The probabilities of the individual phoneme substitutions may be dependent upon the surrounding phonemic context giving rise to correlated sequences of phoneme substitutions (Ch. 1.4).

The probability of each substitution $p(S_i)$ may be dependent upon the preceding and/or following substitutions in the alignment. A more reliable estimate of the substitution probabilities would be calculated by taking the surrounding context of each substitution into account. Each substitution probability may be conditional upon the presence of past substitutions (Left Context (eqn 6.3)), future substitutions (Right Context (eqn 6.4)), or a combination of past and future substitutions (which we call Centre Context (eqn 6.5)) as shown below.

$$p(S_i) = P(S_i|S_1 \ldots S_{i-1}) \tag{6.3}$$

$$p(S_i) = P(S_i|S_{i+1} \ldots S_n) \tag{6.4}$$

$$p(S_i) = P(S_i|S_1 \ldots S_i{-}1, S_{i+1} \ldots S_n) \tag{6.5}$$

It is easy, using elementary rules of probability theory to show that the probability of the sequence $p(S_1 \ldots S_n)$ can be formally decomposed into a product of the conditional probabilities.

$$p(S_1 \ldots S_n) = \prod_{i=1\ldots n} p(S_i|S_1, \ldots, S_{i-1}) \tag{6.6}$$

$$p(S_1 \ldots S_n) = \prod_{i=1\ldots n} p(S_i|S_{i+1}, \ldots, S_n) \tag{6.7}$$

$$\text{but } p(S_1 \ldots S_n) = \prod_{i=1\ldots n} p(S_i|S_1 \ldots S_{i-1}, S_{i+1} \ldots S_n) \tag{6.8}$$

$$\text{iff } p(S_1 \ldots S_{i-1}) \text{ and } p(S_{i+1} \ldots S_n) \text{ are independent} \tag{6.9}$$

Note that centre context probabilities (eqn 6.8) which are conditioned by both left and right contexts cannot in general be used to calculate the sequence probability since the probabilities of the two surrounding sequences $S_1 \ldots S_{i-1}$ and

$S_{i+1} \ldots S_n$ are unlikely to be independent. However, as the sequence probability can be calculated precisely by taking into account either left contexts or right contexts alone this does not appear to be a problem.

Although it is possible to use probabilities conditioned by either left or right contexts it would seem to be more natural and is in practice easier to use the left context sensitive substitutions. These are conditioned by past substitutions which will always be available to calculate the conditional probability whereas right context sensitive substitution probabilities depend upon the presence of future substitutions which can not usually be predicted in advance.

If we are correct in assuming that the context sensitive substitution probabilities are occurring as a result of phonological processes then the amount of preceding left context that conditions a substitution penalty ought to be small. For example, the application of the phonological assimilation rule 's $\mapsto \phi$ / s # _' would cause the probability of an (s.-) substitution to be higher after the sequence of substitutions (s.s)(#.-). As this rule makes no reference to any phonemes before /s #/ we would expect that the substitution probability for (s.-) would be independent of those substitutions that preceded (s.s)(#.-) in the alignment. Even if this assumption is incorrect we would still expect preceding substitutions to have progressively less effect. Eventually the effect of distant substitutions would become insignificant and could be ignored.

Since the range over which context sensitive effects occur is likely to be small it should be possible to calculate the probability of the entire sequence $p(S_1 \ldots S_n)$ as the product of the probabilities of a number of shorter sub-sequences. Although the probability of the substitutions in each sub-sequence may be conditioned by preceding phonemes in the same sub-sequence they should not be conditional to a significant extent upon substitutions in the preceding sub-sequences. In the cases where the a-priori phoneme substitution probabilities are genuinely independent the sub-sequence will consist of a single substitution as before.

These correlated sequences of substitutions correspond to the macro-substitutions (MSubs) that we will be using in the context sensitive DP algorithm. The sequence

probability can now be calculated as the product of the probabilities of the MSubs, provided that each of the MSubs are independent.

$$p(S_1 \ldots S_n) = p(S_1 \ldots S_3)p(S_4) \ldots p(S_{n-5} \ldots S_n) \tag{6.10}$$

The probability of each MSub can also be calculated in an identical manner to that of the entire sequence by taking into account the left context of each substitution using equation 6.6.

The probability of a left context sensitive substitution can be formally decomposed into the product of the a-priori substitution probability and the correlation coefficient $C(S_i \& S_{i-1} \ldots S_1)$ as shown below. The correlation coefficient indicates the degree of correlation between the substitution S_i and its left context $S_{i-1} \ldots S_1$ and is greater than one when the substitution probability is higher in this context and less than one when the substitution probability is lower.

$$\text{Let} \quad C(S_i \& S_{i-1} \ldots S_1) = \frac{p(S_i \ldots S_1)}{p(S_i)p(S_{i-1} \ldots S_1)} \tag{6.11}$$

$$\text{and} \quad p(S_i|S_{i-1} \ldots S_1)p(S_{i-1} \ldots S_1) = p(S_i \ldots S_1) \tag{6.12}$$

$$\text{then} \quad p(S_i|S_{i-1} \ldots S_1) = p(S_i) * C(S_i \& S_{i-1} \ldots S_1) \tag{6.13}$$

The probability of the MSub is calculated using equation 6.6 as the product of the left context sensitive substitution probabilities each of which can be decomposed using equation 6.13. The product of these left context sensitive probabilities can be further decomposed into the product of the a-priori substitution probabilities and correlation coefficients.

$$p(S_1 \ldots S_n) = \prod_{i=1 \ldots n} p(S_i|S_{i-1}, \ldots, S_1) \tag{6.14}$$

$$= \prod_{i=1 \ldots n} p(S_i) * C(S_i \& S_{i-1}, \ldots, S_1) \tag{6.15}$$

$$= \prod_{i=1 \ldots n} p(S_i) * \prod_{i=1 \ldots n} C(S_i \& S_{i-1}, \ldots, S_1) \tag{6.16}$$

$$= C^* * \prod_{i=1 \ldots n} p(S_i) \tag{6.17}$$

$$D(S_1 \ldots S_n) = \log C^* + \sum_{i=1\ldots n} \log p(S_i) \quad (6.18)$$

$$\text{where } C^* = \prod_{i=1\ldots n} C(S_i \& S_{i-1} \ldots S_1) \quad (6.19)$$

Note that $\sum_{i=1\ldots n} \log p(S_i)$ is precisely what the standard DP algorithm originally calculated for the sequence probability (equation 6.1). C^* is a constant correction factor that can be calculated in advance for each MSub. The correction factor takes into account the effects of preceding substitutions on each substitution in the MSub.

The decomposition leading to equation 6.17 shows that the probability of a correlated sequence of substitutions (i.e. a MSub) can be calculated using the standard a-priori DP substitution probabilities along with a single correction factor C^* which is simply the product of the individual correlation coefficients. This is significantly more efficient than using equation 6.6 to calculate the left context sensitive probabilities for each of the substitutions as we have in effect compensated for many conditional probabilities within a Msub with the use of a single correction factor C^*.

As an example consider a sequence of four substitutions $(S_1 \ldots S_4)$ with each substitution S_i having an a-priori probability of 0.1. If the conditional probabilities of these substitutions are as follows the probability of the entire sequence can be calculated as the product of the four conditional probabilities using equation 6.6.

$$\begin{aligned} p(S_2|S_1) &= 0.2 \\ p(S_3|S_1 \ldots S_2) &= 0.3 \\ p(S_4|S_1 \ldots S_3) &= 0.4 \end{aligned}$$

$$\begin{aligned} p(S_1 \ldots S_4) &= p(S_1) * p(S_2|S_1) * p(S_3|S_1 S_2) * p(S_4|S_1 S_2 S_3 S_4) \\ &= 0.00024 \end{aligned}$$

By rearranging equations 6.11 to 6.13 we can calculate the correlation coefficients C as $p(A|B)/p(A)$ which gives values for the three coefficients as 2, 3 and 4

respectively. This gives a correction factor C^* for the entire sequence of 24 (eqn 6.19). The probability of the MSub is calculated using equation 6.17 as follows.

$$\begin{aligned} p(S_1 \ldots S_4) &= C^* * \prod_{i=1\ldots4} p(S_i) \\ &= 24 * 0.00001 \\ &= 0.00024 \end{aligned}$$

Both calculations give the same results although the latter involved simply multiplying four independent substitution probabilities and a single correction factor whilst the former involved calculating three conditional probabilities.

6.1.1 Dynamic Programming with Macro-Substitutions

The context sensitive DP algorithm (described in detail in the next chapter) uses the Msubs to calculate an optimal alignment which consists of a sequence of standard independent DP substitutions (X.Y) and MSubs [(A.B) ... (X.Y)]. The context sensitive DP alignments are calculated in two separate stages. At the beginning of each time interval the partial alignments are propagated as usual by the DP alignment algorithm based upon the assumption that the substitution probabilities are independent. In the second stage correlated sequences of phoneme substitutions corresponding to MSubs are identified. The probability of these correlated sequences will not have been calculated properly by the DP algorithm during the first stage so it will be necessary to recalculate the probabilities of these sequences. The probability of each MSub is recalculated using the standard context free DP algorithm in an almost identical way to that of the original context free DP alignments. The final score of the DP alignment is obtained by adding in the correction factor of the MSub.

There are likely to be a large number of MSubs, in part due to the presence of numerous phonological rules. It is therefore convenient to use a tree data structure (e.g. fig. 6–3) to store the entire set of MSubs since this provides a compact and computationally efficient representation. Each (non depth-1) node of the MSub

tree corresponds to a single MSub and MSubs which share the same initial sequence of phoneme substitutions have the same parent node in the tree. Applicable MSubs are identified by matching the substitutions stored on successively deeper nodes of the tree with the contents of the template's reference frames and the contents of the input lattice. However, the order in which MSubs are stored in the tree can have significant implications on the efficiency with which applicable MSubs can be identified and implemented.

In speech recognition systems using hidden markov models and finite state transducers it is usual for the additional paths arising from the application of phonological rules to be propagated in parallel with the base-form paths. The same techniques (which we call forward scanning) can also be used to time synchronously propagate paths through the MSub tree in parallel with the standard context free DP paths. However, as we shall see, significant savings can be achieved by processing the MSubs in reverse (i.e. reverse scanning). Instead of incrementally propagating paths in parallel with the context free DP paths we choose, for the reasons outlined below, to identify potential MSubs in reverse by backtracking through the template reference frames and previous lattice contents before we begin to calculate the corrected alignment scores.

6.1.2 Forward Scanning

The DP alignment scores of the MSubs in the tree can be extended time synchronously along with the other paths currently being extended by the DP alignment algorithm. This would involve calculating DP penalties for those substitutions (X.Y) stored on depth one and subsequent nodes whose template 'X' and input 'Y' components were compatible with the contents of the current template reference frame and lattice segment.

The accumulated distance scores and backtracking pointers of each compatible substitution will have to be stored on the corresponding nodes of the tree so that these values are available when required to propagate partial paths in the next time interval. The paths stored on depth n nodes would be propagated during

the next time interval and extended onto the depth $n + 1$ child nodes that also had compatible phonemes. This process would be repeated with the arrival of each new lattice segment. The paths would be extended time synchronously onto successively deeper nodes of the tree until either there were no more compatible child nodes or a leaf node was reached.

In figure 6–1 the DP path would follow the bold line since this is the only branch whose substitutions (X.Y) are compatible with the template 'X' and input 'Y' phonemes when matching begins at the indicated template reference frame. Whenever a node corresponding to an MSub is reached (e.g. (z.z)) the correction factor stored on the node can be used to give a more accurate estimate of the actual path score. The new estimate of the DP alignment score of the substitutions along the branch can then be compared with the other context free paths that have been calculated by the DP alignment algorithm. If the corrected context sensitive DP alignment score was lower than the context free DP alignment score the path would be transferred from the branch into the DP matrix where it could be extended along with the other paths during future time intervals.

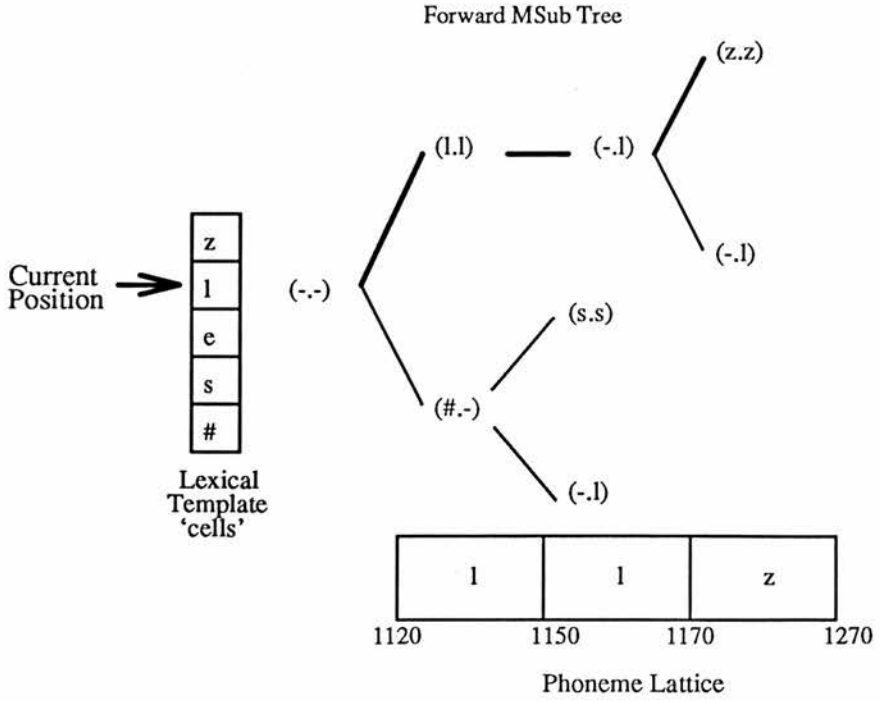


Figure 6–1: Using MSub trees in the forward direction

Unfortunately there are a number of potentially serious disadvantages with this apparently straight forward approach. First of all the identification and implementation of Msubs in the forward direction is non-deterministic. We cannot know in advance whether a length N Msub which has been successfully matched so far will in fact turn out to be part of longer length $N+1$ MSub that should be applied instead. For example, in figure 6–1 the MSub [(l.l)(-.l)] could be applied in the alignment of 'cells' between 1120 and 1170mS. However, we cannot be sure that this MSub should be used to replace the corresponding context free path fragment in the DP matrix until we have seen the contents of the following lattice segments. In this example the MSub would have been extended in the next time interval since the substitution (z.z) on the next node is consistent with the next phonemes in the template and input segment. The correct choice would have been to wait and then replace the original path fragment with the alignment of the longer MSub [(l.l)(-.l)(z.z)].

The non-deterministic nature of this approach means that a significant amount of temporary storage would be required to store the accumulated distance scores and backtracking pointers of the substitutions on nodes that have been successfully matched so far. This information needs to be stored on every matched node in the tree since the distance scores and backtracking pointers may have to be copied from the branch into the DP matrix if the corrected alignment score of the MSub is sufficiently low. The information will have to be stored until the MSubs corresponding to the longest branches in the tree have been successfully applied or ruled out due to incompatible phonemes in the reference or input segments.

We would also have to create a large number of instances of the MSub trees. A separate instance of the MSub tree would be needed for every phoneme of each template in the lexicon and a completely new set of these trees would have to be created whenever a new lattice segment was encountered. The separate instances would be required to store the different accumulated distance scores and backtracking pointers that would be obtained whenever the matching process began at a different template reference phoneme or lattice segment.

For example, in figure 6–1, the depth 1 node in the tree corresponding to

the substitution (l.l) could be used to propagate paths which consume the input phoneme /l/ starting at 1120mS and the fourth phoneme of 'cells' as well as paths that consume the /l/ input phoneme beginning at 1150mS. The same node would also be used when propagating paths in any other template that contains one or more /l/ phonemes. It will be be necessary to have a separate instance of this and other nodes in the MSub tree so as to avoid avoid confusing the accumulated distance scores and backtracking pointers of nodes whose substitutions were matched with template or input phonemes.

This apparently straightforward approach therefore requires a great deal of computation and run-time storage. The run-time overheads are at least as large as those which would have been required if the equivalent set of rules had been simply used to pre-compile the lexicon. There would therefore seem to be little advantage to be gained in this respect by applying MSubs in this way at run-time.

6.1.3 Reverse Scanning

Potential MSubs could also be identified by scanning the path fragments in reverse i.e. starting with the most recent substitution in the MSubs and then working backwards, checking the plausibility of successively earlier substitutions. In order to achieve this we will need to construct a reverse tree as shown in fig. 6–2 (the algorithm which achieves this is described later). Each node in the reverse tree contains the substitution in the sequence which precedes the substitution stored on its parent node. For example, the branch (z.z)(-.l)(l.l) corresponds to the substitution (z.z) preceded by (-.l) preceded by (l.l) and is therefore equivalent to the MSub [(l.l)(-.l)(z.z)] in the previous tree. The reverse tree in fig. 6–2 is actually the tree which would be generated by inverting the tree in fig. 6–1. Both of these trees contain the same set of MSubs.

We can also identify potential MSubs in reverse by matching the substitutions on the nodes of the tree with the corresponding phonemes in the input lattice and template reference frames. Each node in the reverse tree contains substitutions that are matched with phonemes in preceding lattice segments and template refer-

ence frames. The substitutions on successively deeper nodes are matched (but not scored) with progressively earlier lattice segments and template reference frames. As before the branch (z.z)(-.l)(l.l) corresponding to the MSub [(l.l)(-.l)(z.z)] can be matched against the lattice segments and reference frames.

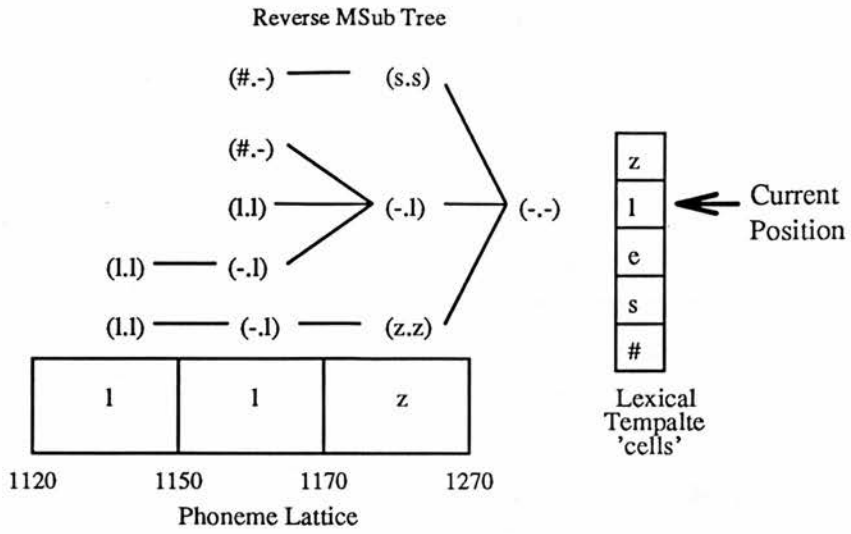


Figure 6–2: Scanning N-gram trees in the reverse direction

Since all of the preceding lattice segments and template reference frames are already known it is now possible to unambiguously determine the longest MSub which should be applied. The identification of potential MSubs is therefore completely deterministic and the previous problem which involved choosing between applying the MSub [(l.l)(-.l)] or waiting to see if the longer MSub [(l.l)(-.l)(z.z)] should be applied no longer arises. This problem can be solved at the time at which the MSub should be applied when matching each of the substitutions in reverse.

Since the identification of potential MSubs is deterministic we no longer have to store intermediate data about potential MSubs which may or may not turn out to be applicable at some future time. We can always determine the longest MSub that is applicable at the current time by looking at progressively earlier substitutions. Once the longest applicable MSub has been identified the DP score of the path fragment is calculated and then modified according to the correction factor of the MSub. Although we still have to store details such as accumulated

distance scores and backtracking pointers on the nodes of the tree whilst the DP alignments are being propagated along successfully matched branches these can be discarded once the MSub has been applied. Since all of the temporary data is discarded after each MSub has been applied the same tree can be repeatedly reused with each of phonemes in the templates and also for successive lattice segments. This drastically reduces the amount of run-time storage that is required.

6.2 Techniques For Automatically Extracting Macro-Substitutions

It is possible to automatically extract Msubs by looking for significantly correlated sequences of phoneme substitutions within alignments that have been previously generated by the DP alignment algorithm. The techniques which are used for detecting and extracting MSubs are based upon a program called Treelab [Rohwer].This is a statistical modeling program which constructs a tree of N-grams from a body of data consisting of a sequence of arbitrary symbols. An N-gram is simply a sequence of N symbols and these are stored in the tree along with tallies which record how often the sequences were observed in the corpus. The probabilities of individual symbols (1-grams) and sequences of symbols (N-grams) as well as the conditional probability $p(X|Y)$ of a symbol X given a preceding sequence Y can all be calculated directly from the N-gram tallies.

Treelab uses a convenient and compact data structure called a 'treegram' to store the numerous N-grams that are generated from the corpus. A treegram (e.g. fig. 6–3 is a tree of N-grams where each node of the tree identifies a unique N-gram. The root node of the tree has a unique symbol and a tally which records the total number of symbols in the corpus. 1-grams are stored on depth one nodes which contain a symbol identifying the 1-gram and a tally which records how often the symbol was observed in the corpus. Likewise, 2-grams, 3-grams, and so on, are stored on successively deeper nodes which contain the last symbol of the sequence and a tally that records the number of times the sequence was observed. Depth

n nodes occur as the children of depth $n-1$ nodes. The depth $n-1$ nodes represent the sequences that provide the left contexts for the symbols stored on the depth n nodes.

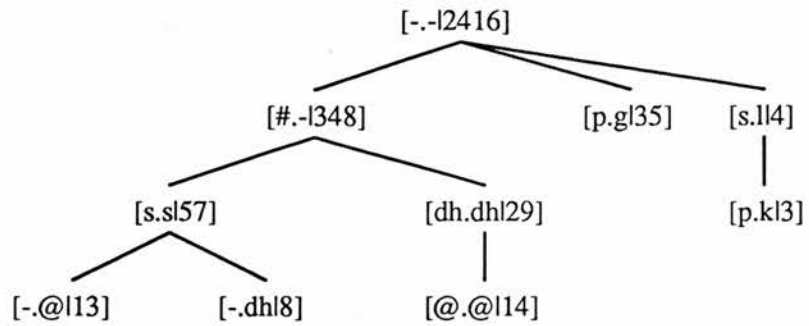


Figure 6–3: A small fragment of a 'forward' tree produced by Treelab

Figure 6–3 shows a small fragment of a tree that was constructed by treelab from our training corpus which consists 60 DP alignments of various utterances from the cytology domain. The tree consists of a number of nodes each of which represent an individual phoneme substitution. Each node has a single parent which corresponds to the preceding substitution in the sequence and may have a number of child nodes depending upon the range of substitutions that can follow. The tally of each node records the number of times the sequence of phoneme substitutions along the branch from the root to the node were observed. The root node [-.-] records the total number of substitutions in the corpus.

The independent substitution probabilities from which the a-priori DP substitution penalties were derived can be simply calculated from the ratio of the depth one node tallies and the root tally. Likewise the probabilities of longer sequences can be calculated from the ratio of their depth N node tallies and the root tally. Conditional substitution probabilities which depend upon the preceding left context are calculated as the ratio of the node tally to the tally of its parent node.

Inevitably, because of finite corpus sizes there will be uncertainties in the tally counts and the probabilities which are calculated from them. It has been shown [Rohwer 87] that a reasonable estimate of the uncertainty in the tally counts is equal to $\sqrt{tally}$. Long sequences typically have much lower tallies than shorter

sequences since the tally of a sequence can't be greater than the tallies of any of its sub-sequences. The uncertainties in the subsequent probability calculations can therefore be assumed to be dominated by uncertainty in the tally of the longest sequence.

6.2.1 Tree Construction

Treelab is used to constructs an N-gram tree from a corpus of phoneme alignments. The corpus consists of 60 alignments which were obtained from the DP algorithm during training. These are the optimal alignments between the segments in the phoneme lattice and the correct sequence of templates and can therefore be assumed to contain the correct sequences of phoneme substitutions. Word boundary deletions (#.-) are treated like any other phoneme substitution during tree construction whereas sentence boundaries are used to define separator points in the corpus. Seperator points are points where one sequence ends and another begins. It seems sensible to break the substitution sequences at these points since the sentences were spoken in isolation. It is therefore unreasonable to expect the sequences of phoneme substitutions at the end of one sentence will be correlated with those at the start of the next in any useful way.

The tree is built up gradually, adding branches of successively longer length during each pass over the set of training alignments. At the end of the first pass the root node is created and given a unique symbol [-.-] and a tally which is equal to the total number of substitutions in the corpus. On the next pass over the corpus branches of length one are added. During this pass a new node is added for each type of phoneme substitution and given a tally that records how often the substitution occurred within the corpus. Branches of length two and longer are then added on subsequent passes over the corpus. The leaf nodes on branches of length N have tallies which record the number of occurrences of the sequence of N phoneme substitutions stored on the nodes of the branch from the root to the leaf node.

In order to create branches for all sequences of length N Treelab processes each of the alignments in turn and extracts sequences of length N beginning at each of the $n = 1 \ldots L - N$ substitutions in the alignment, where L is the total number of substitutions in the alignment. At each step n the N-gram consisting of the following N substitutions is extracted and matched, depth first, with existing branches in the tree. The first substitution of the sequence is matched with depth one nodes, the second with the children of the matching node and so on until the entire sequence has been matched or no matching node can be found amongst the children of the last node to have been successfully matched.

If the entire sequence is successfully matched with a branch in the tree the tally of the leaf node, which records how often the sequence of substitutions along the branch has been observed so far, is incremented. However, if this is the first occurrence of the length N sequence the depth first search will fail to find a branch which completely matches the sequence. If matching fails at the final substitution of the sequence a new child node is grown from the last node to be successfully matched and is given an initial tally of 1 and a symbol which corresponds to the final substitution of the length N sequence.

After each pass over the corpus, each of the new depth N leaf nodes are examined and those which have a tally less than a pre-specified MINTALLY (typically in the range $1 \ldots 10$) are pruned from the tree. The pruning process removes nodes from the tree which represent sequences that have not occurred often enough to be able to provide reliable tally counts (for example the uncertainty in a tally of 10 is 33% ($\sqrt{10}$). This process of growing and pruning is repeated for successively longer sequences and branches until either no new nodes are created or all new nodes are subsequently pruned.

It is possible that matching will fail before the final substitution of a length N sequence if the length $N - 1$ sub-sequence occurred less than MINTALLY times in preceding passes and was subsequently pruned. When this occurs there will no node in the tree from which a child can be grown to represent the length N sequence. However this does not present a problem since it is impossible for the length N sequence to occur more often than any of its sub sequences. This means

that the length N sequence must also occur less than MINTALLY times if the length $N-1$ sub-sequence occurred less than MINTALLY times and was pruned. There is therefore no reason to try a construct a branch to represent the length N sequence since even if it were possible to create one it would be pruned at the end of the pass.

6.2.2 Tree Inversion

The next stage is to invert the tree produced by Treelab. This process creates a tree which stores the sequences in reverse where successively deeper nodes in the tree correspond to progressively earlier substitutions in the sequence. The original tree (fig. 6–4(a)) and the inverted tree (fig. 6–4(b)) have branches which represent identical sequences (e.g. (l.l)(-.l)(z.z) which is also represented as (z.z)(-.l)(l.l) in the inverted tree).

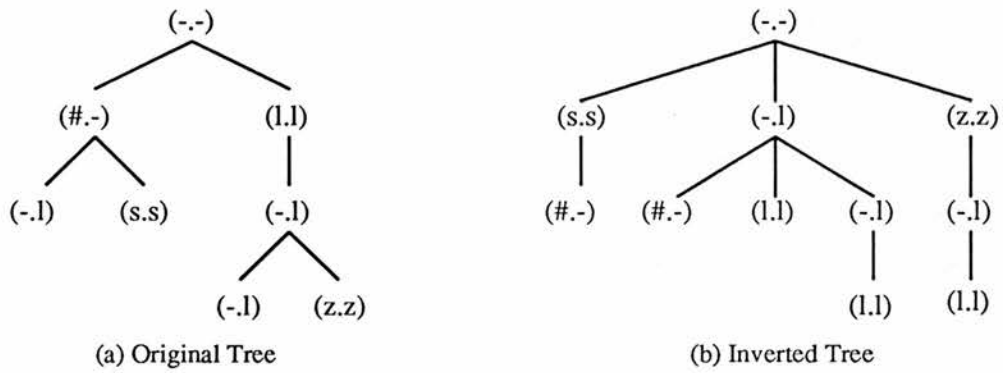


Figure 6–4: An example of an original and inverted tree

Although the structure of the two trees is different they still contain precisely the same sequences which each have identical tallies in both trees. Obviously if the sequence (#.-)(s.s) occurred 57 times when scanning in the forward direction, the same reversed sequence (s.s)(#.-) would also have been observed 57 times if we had started scanning in the reverse direction. The obvious way to calculate the inverted tree would be to run Treelab over the corpus in reverse. However, reading files in reverse is generally neither a straightforward nor efficient operation. Fortunately

it is relatively simple to invert the tree calculated in the normal forward direction to achieve the same effect.

The procedure **invert** which recursively constructs the inverted tree is outlined in fig. 6–5. A new node `rev_root` is initially created for the reversed tree which has the same substitution and tally as the root node `fwd_root` of the initial tree. Each of the nodes in the original tree are then visited during a recursive depth first descent. As each `child_node` is visited the sequence represented by the nodes along the branch from the `fwd_root` to the `child_node` is transferred, in reverse order, into the reverse tree. This is achieved by ascending the branch from the `child_node` to the `fwd_root` whilst descending along the corresponding branch in the `rev_tree`. The phoneme substitutions along the nodes (`fwd_nodes`) of the branch are used to find the equivalent nodes (`rev_nodes`) when descending along the corresponding branch in the reverse tree.

```
invert(fwd_root, rev_root)
  for child_node = 1 .... children(root1)

    rev_node = rev_root;
    for fwd_node = subtree .... fwd_root
      rev_node = eqv_child(rev_node, fwd_node)
    tally(rev_node) = tally(child_node)

    invert(sub_tree, rev_root)
```

Figure 6–5: Tree Inversion Algorithm

If the branch is not present in reverse tree it will not be possible to match the initial substitution (i.e. depth 1 node) of the branch with the outstanding leaf nodes in the reverse tree. If this occurs a matching leaf node is created and given a tally equal to the tally of the `child_node` from which the matching process started (i.e the tally of the original sequence). This process is repeated recursively for all of the nodes in the initial tree and will eventually create the inverted tree.

6.2.3 Tree Decomposition

The inverted tree contains nodes which represent all sequences that were observed in the corpus at least MINTALLY times. However we are only interested in those sequences which contain correlated sequences of phoneme substitutions i.e. MSubs. Many of the branches will represent sequences that are not significantly correlated. Even when a branch does represent a significantly correlated sequence it is still possible that the sequence is in fact made up of two or more uncorrelated sub-sequences. Each of these sub-sequences could then be used to calculate the probability of the entire sequence using eqn. 6.10. The aim of tree decomposition is to decompose the large tree originally produced by Treelab into a smaller tree which contains the minimal number of short, significantly correlated sequences. These shorter sequences are more likely to correspond to the phonological phenomena that we are interested in and, as the tallies are likely to be higher, the uncertainties in the subsequent calculations will be reduced.

A sequence $S_1 \ldots S_n$ is decomposed into two shorter sub-sequences $S_1 \ldots S_i$ and $S_{i+1} \ldots S_n$ if the product of the probabilities of the sub-sequences is equal to the probability of the entire sequence, i.e. if the two sub-sequences are uncorrelated. In practice there will be uncertainties in each of the probability estimates which can be used to determine whether any difference between the probability of the entire sequence and the product of its sub-sequences is statistically significant. The uncertainty between the probability estimates will be dominated by that of the longer sequence $p(S_1 \ldots S_n)$ which will have the lowest tally. The sequence probabilities and the estimates of their uncertainties are used to construct a minimal deterministic (i.e. reversed) Dmin-tree as follows.

A root node is created for the Dmin-tree which has the same tally as the root node in the original tree (Dmax-tree). All of the depth 1 nodes are then copied from the Dmax-tree to the Dmin-tree. Then, descending through the Dmax-tree breadth first all branches of length n are examined, starting with $n = 2$. The probability of the sequence S represented by the last node of each length n branch and the estimate of its uncertainty are calculated from the node and root tallies.

The probability of the sequence S is then estimated using the sub-sequences represented by the nodes in the Dmin-tree. The initial estimate of the probability starts at 1.0 and is then updated during a recursive depth first descent through the Dmin-tree tree which matches symbols in the sequence S against successively deeper nodes in the tree. When a leaf node is reached the current estimate of the sequence probability is updated by multiplying the current probability estimate by the probability stored on the leaf node. The process is then repeated, starting once again from the root node of the Dmin-tree, until all of the symbols in the sequence S have been successfully matched. The final probability estimate is then calculated by multiplying by the probability value stored on the last node that was matched.

If the probability estimate is sufficiently close to that which was originally calculated from the Dmax-tree then there is no need to add the branch to the Dmin-tree since we will have shown shown the probability of the sequence can be calculated with sufficient accuracy using the existing sequences in the Dmin-tree. However, if the probabilities are significantly different then the sequence must be copied in its entirety into the Dmin-tree so that the correct sequence probability can be obtained in the future.

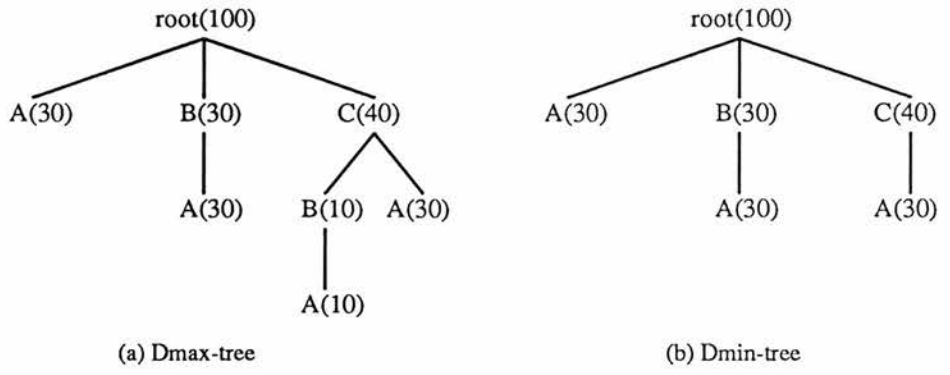


Figure 6–6: (a) shows a simple Dmax-tree that might have been produced by Treelab and (b) is the Dmin-tree that will be produced by the decomposition algorithm. Both trees give similar probability estimates for sequences in the Dmax-tree

It should be clear that as the Dmin-tree is constructed breadth first, all longer

sequences will be decomposed into a sequence of shorter sub-sequences if possible and therefore the Dmin-tree will contain the minimal set of shortest sequences from which the probability of all other longer sequences can be estimated without significant error. These short branches will have the highest tallies and hence the lowest uncertainties in their probability estimates. The following example is used to illustrate this process.

In this example we are going to decompose branches of length 3. The branch in the Dmax-tree (fig. 6–6(a)) representing the sequence ABC has a probability of 0.1 (i.e. 10/100). Another estimate for the probability of this sequence is calculated as described previously by matching the branch with existing nodes in the Dmin-tree (fig. 6–6(b)). The initial probability estimate p is set to 1.0 and then we begin matching nodes in the tree. We match the 'C' of the sequence 'ABC' but we can go no further so the probability estimate is updated from this node ($p = p * 0.4$). Matching begins again, starting once again at the root and this time we are able to match the B preceded by the A. This completes the matching of the sequence ABC and the final estimate is updated from the last node matched ($p = p * 0.3$). This gives an estimate for the probability of the sequence ABC of 0.12 which was calculated as $p(AB)p(C)$. This estimate is within the uncertainty, $\frac{\sqrt{10}}{100} = 0.3$, of the original probability which was calculated to be 0.1. The sequence ABC therefore does not have to be added to Dmin-tree.

6.2.4 Calculating Correction Factors

The final step is to calculate the correction factor for each of the branches in the Dmin-tree. Recall that the correction factor of each MSub is equal to the product of the correlation coefficients between each substitution and its preceding left context (eqn 6.19). The correlation coefficients can be simply calculated from the node tallies stored in the tree as follows where $n(S_1 \ldots S_i)$ is the tally of a depth i branch, $n(S_1 \ldots S_{i-1})$ is the tally of its parent node, $n(S_i)$ is the tally of the depth one node which records the number of occurrences of substitution S_i and $n(S)$ is the tally of the root node which records the total number of substitutions.

$$C(A\&B) = \frac{p(A|B)}{p(A)} \tag{6.20}$$

$$= \frac{p(A\&B)}{p(A)p(B)} \tag{6.21}$$

$$= \frac{p(S_i\&S_1 \ldots S_i)}{p(S_1 \ldots S_{i-1})p(S_i)} \tag{6.22}$$

$$C(A\&B) = \frac{n(S)n(S_1 \ldots S_i)}{n(S_1 \ldots S_{i-1})n(S_i)} \tag{6.23}$$

Values for the correlation coefficients are calculated for each node in the tree during a depth first descent of the tree. The MSub correction factors C^* are then calculated by once again descending the tree and storing the product of the previously calculated correlation coefficients on each node. The correction factor of each MSub of length i is therefore stored on the corresponding depth i node and is equal to the product of the correlation coefficients along the branch from node 1 to i.

Chapter 7

Dynamic Programming with Macro-Substitutions

The Dmin-tree that is produced as the output of the tree decomposition stage described in the previous chapter contains the minimal set of all reliably correlated phoneme substitution sequences. These will include positively correlated sequences which occurred more often than would have been expected from the individual a-priori substitution probabilities as well as negatively correlated sequences which occurred less often than would have been expected. Branches in the tree will therefore represent sequences of phoneme substitutions whose probability will not be accurately calculated by the standard DP alignment algorithm which assumes that the phoneme substitution probabilities are independent.

Errors in the probability calculations are likely to be reflected as errors in the optimal alignments since the determination of the optimal alignment is based upon the probabilities of the individual phoneme substitutions. The tree of correlated phoneme substitution sequences (MSubs) is used to identify those path fragments whose probability will have been incorrectly calculated by the DP algorithm and recalculate context sensitive scores which take into account the conditional probabilities of the individual phoneme substitutions.

Different techniques are required to implement positively correlated macro-substitutions (P-MSubs) and negatively correlated macro-substitutions (N-MSubs). The DP alignment algorithm obtains the optimal alignment by finding the low-

est scoring extension of every path at each time interval. However the standard context-free DP alignment algorithm will calculate scores for partial alignments containing P-MSubs which are too high (i.e. whose probabilities are too low) and scores for partial alignments containing N-MSubs which are too low (i.e. whose probabilities are too high). This means that non-optimal paths may be chosen instead of paths containing P-MSubs which should have scored lower and paths containing N-MSubs which should have scored higher may be chosen instead of the genuinely optimal paths.

As a result path fragments corresponding to P-MSubs may no longer be present in the DP matrix (having been displaced by lower scoring paths) and need to be rehypothesised with their corrected alignment scores whilst path fragments corresponding to N-MSubs should be prevented from displacing other paths in the DP matrix until their corrected alignment scores have been calculated. This can not be achieved by rehypothesising the path fragments with corrected scores since these fragments will always have a higher score than the incorrectly scored fragments we would want to replace. The path fragments with corrected scores would always be (incorrectly) displaced by the original alignments which have scores that are too low.

Positively correlated macro-substitutions (P-MSubs) and negatively correlated macro-substitutions (N-MSubs) have to be treated in different ways. It is therefore convenient to break up the original Dmin-tree tree which contains both positively correlated P-MSubs and negatively correlated N-MSubs into two smaller trees which contain just P-MSubs or just N-MSubs. This allows both kinds of MSubs to be treated independently and efficiently by appropriately specialised routines. Since each of these trees is handled independently we are also able to selectively choose at run-time the kind of MSubs that we wish to use.

In addition to the two MSub trees we also derive a third pre-wb tree from the original Dmin-tree. This tree contains branches which represent those subsequences which can precede word boundary deletions (#.-) in MSubs. As we saw in the discussion of the chart parser MSubs which span word boundaries pose special problems. In this situation the choice of optimal templates whose

alignments end at the current time interval may be dependent upon the sequences of substitutions in the alignments of templates which will begin in the following time interval. The pre-wb tree is used to identify those templates whose presence in the list of optimal templates is dependent upon the alignments of future templates and also provides values for the Ltags and Rtags which are used by the chart parser.

The original context-free dynamic program remains at the heart of the context sensitive dynamic programming algorithm. This still operates in the usual way (as described in Chapter 3) and generates the optimal extensions of paths based upon the assumption that the substitution penalties are independent. The branches in the P-MSub and N-MSub trees are used to identify sequences of substitutions where this assumption is incorrect and to provide corrected alignment scores which may prevent paths from being added to the DP matrix in the case of N-MSubs or cause missing path fragments to be inserted in the case of P-MSubs.

The actual implementation of a macro substitution occurs over a number of stages each of which attempts to reduce the number of potential MSubs that have to be considered in the following, usually more time consuming stages. The first stage which is applied to each of the three trees consists of pre-match pruning (Ch. 7.3). This takes place at the beginning of each time interval and produces substantial reductions in the size of the trees which are then passed on to the specialised routines which handle word boundaries (Ch. 7.2), P-MSubs (Ch. 7.4) and N-MSubs (Ch. 7.5).

7.1 Identifying Applicable Macro Substitutions

In the discussion between the relative merits of forward and backward scanning (Ch. 6.1.1) we saw that it was possible to deterministically implement MSubs by scanning the trees in reverse and matching the branch nodes with progressively earlier substitutions which can always be made available. In the alternative forward scanning technique paths were propagated along the branches time synchronously, in parallel with those generated by the standard DP algorithm. A

great deal of computation and storage would have been wasted in this approach due to the propagation of paths which were never used because of the lack of suitable phonemes in future template frames and/or lattice segments. The deterministic nature of the reverse scanning approach means that the processes of identifying applicable MSubs and calculating their alignment scores can be usefully separated. It is possible to always identify applicable MSubs in advance by looking at successively earlier substitutions and then only perform the relatively time consuming alignment calculations on MSubs which have been determined to be applicable.

In this section we will give a very brief overview of the processes involved in identifying and applying macro substitutions. A more detailed explanation of the actual algorithms is given in the following sections of this chapter. In order to illustrate the individual stages we will consider a branch (z.z)(-.l)(l.l) in an MSub tree (either positive or negative) which corresponds to the macro substitution [(l.l)(-.l)(z.z)]. Since the sequence of phoneme substitutions is correlated its alignment score will be incorrectly calculated by the DP algorithm. We must therefore be able to identify the situations in which path fragments corresponding to this and other MSubs could occur and either rehypothesise the path with the corrected score in the case of P-MSubs or correct the score before the path is added into the DP matrix in the case of N-MSubs.

It is only possible for a path fragment such as (l.l)(-.l)(z.z) to occur in the DP alignment of a template when the template component 'X' and input component 'Y' of each substitution '(X.Y)' matches the contents of successive template reference frames and input lattice segments. The path fragment (l.l)(-.l)(z.z) can only be present in the DP alignment matrix when the template reference frames contains the phoneme sequence /l z/ and the lattice segments contain the phoneme sequence /l l z/.

If any of the template reference frames or lattice segments do not contain phonemes which match those in the corresponding substitutions of the MSub it will have been impossible for the DP algorithm to produce the path fragment when generating the alignment of the template. Since the DP algorithm could not

have produced the path fragment under these circumstances any effort to try and calculate a corrected DP alignment scores for this path will be clearly wasted.

An MSub [(X.l)(Y.l)(Z.z)] (where X,Y,Z are variables which can represent any phoneme) will never be applied if any of the input components /l l z/ are not present in the corresponding lattice segment, irrespective of the values of the template components. Likewise the MSub [(l.X)(-.Y)(z.Z)] will never be applied if any of the template components /l z/ are not present in the corresponding reference frame (the null phoneme /-/ has zero duration and can be ignored during matching), irrespective of the values of the input components. This means that it is possible to identify applicable MSub in two independent stages which match the template and input components independently.

We could initially match all of the template components of an MSub and then go back and match the input components of those MSubs whose substitutions had compatible template components and vice-versa. However the template components have to be matched against the reference frames of each template in the lexicon and this is a time consuming process. A significant amount of computation can be saved by removing those MSub which have incompatible input components before they are matched against the reference frames of each template. Since the MSubs are stored in a tree structure removing incompatible MSubs is achieved by pruning the leaf nodes from the tree which represent those MSubs. This process, which we call 'pre-match pruning' will be described in more detail in the following section.

Pre-match pruning removes all MSubs whose substitutions contain at least one incompatible input component. In the second matching stage we can ignore the input components (since these have already been checked) and concentrate on matching the template components of the remaining MSubs with the reference frames of each template in the lexicon. If the template components are also found to be compatible with the reference frames of a given template (in this example 'cells' or 'reveals' would enable successful matches) then we will have identified an applicable MSub.

Note however that the successful identification of an MSub does not necessarily

mean that the path fragment actually forms part of the optimal alignment or that it is even present in the DP matrix. We have in fact only checked that the template and input components of each substitution are available. In checking the substitutions of the MSub [(l.l)(-.l)(z.z)] we only ensure that the corresponding template and reference phonemes are available to perform the substitutions, we have not determined whether the actual substitutions are present in the DP matrix.

This is actually all that is required to identify applicable P-MSubs since in this case the positively correlated MSubs describe path fragments that will have an abnormally high score and which may have been displaced from the DP matrix by other path fragments which apparently score lower. We therefore want to identify paths which could (should) have been in the matrix but which are possibly not. However, additional computation will be needed to identify applicable N-MSubs. These correspond to negatively correlated sequences that should be prevented from occurring with abnormally low scores. In this case we must also check that the substitutions are actually present in the DP matrix in order to identify N-MSubs which are in danger of occurring. The slightly different identification requirements are only manifested in the second stage matching algorithms which are described in sections Ch. 7.4 and Ch. 7.5 and not in pre-match pruning.

7.2 Word Boundary Spanning Macro Substitutions

It is a well known fact that phonological rules which span word boundaries are particularly difficult to implement in practice and in this respect macro substitutions are no exception. One solution to this problem was devised by T. Vintysyuk [Vintsyuk 71] and has since been used in virtually all connected speech recognition systems which incorporate phonological rules that span word boundaries. His solution was to pre-compile the grammar and phonological knowledge into the lexicon. The lexicon which is produced is a complex network of arcs between template phonemes where any path through the network corresponds to a particular

pronunciation of a grammatical sentence. All valid pronunciations of templates are represented by arcs through the network. Additional arcs are also used to join syntactically valid words. Our example phrases 'fast speech' and 'that speech' would be represented as shown in fig. 7–1. The bold black line is in many ways equivalent to our MSub [(s.s)(t.-)(#.-)(s.-)] which allows the /t/ and /s/ phonemes to be ignored (deleted) in a pronunciation of 'fast speech'.

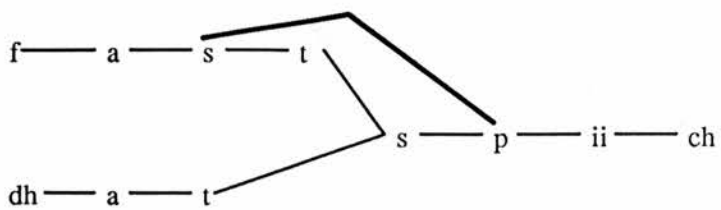


Figure 7–1: This network allows two possible pronunciations of the phrase 'fast speech'. Taking the alternative pronunciation provided by the bold path corresponds to the application of the MSub [(s.s)(t.-)(#.-)(s.-)]

As we have seen in chapter 2 this approach rapidly leads to excessively large and complex lexicons and this was one of the reasons which led us to look for alternative techniques. Since we cannot precompile the cross word boundary rules into the lexicon we must expect that many of the decisions which are made about the locally optimal templates from which new alignments are propagated will turn out to be incorrect. We should therefore try to take measures which will allow the DP algorithm to recover from any errors as and when they occur. Instead of making a hard decision, based upon possibly incorrect assumptions, we choose to create a tree containing all of the potentially optimal templates. This will include all of the templates that may have to be in the list of the genuinely optimal templates in order to allow cross word boundary MSubs to be applied in the following template when determining the optimal alignments.

7.2.1 The problem

At the end of each time interval the DP algorithm needs to be able choose the lowest scoring templates in each of the syntactic categories so that these can be passed

onto the parsing component. The parser uses the templates to propagate partial parses and determines the set of syntactic categories containing templates whose alignments can be concatenated. However, it may turn out to be the case that the alignment score of a future template is dependent upon the application of a word boundary spanning MSub which requires a particular sequence of substitutions to be present in the alignment of the preceding template.

For example, at some time interval the DP alignment algorithm might obtain alignments for the templates 'fast' and 'that' as shown in figure 7–2. Information about these templates would be sent to the chart parser as described previously. The chart parser would in turn return messages stating that either of these alignments could be concatenated with new alignments in templates of category 'n' such as 'speech'. In attempting to calculate the optimal alignment the DP algorithm will naturally propagate the new alignment of 'speech' from the lowest scoring template i.e. from 'that'. The accumulated distance scores and backtracking pointers of the new alignment will be based upon and refer back to the template 'that'. This will lead to the alignment as shown for the phrase 'that speech' which, in the absence of cross word boundary spanning macro substitutions, is in fact optimal.

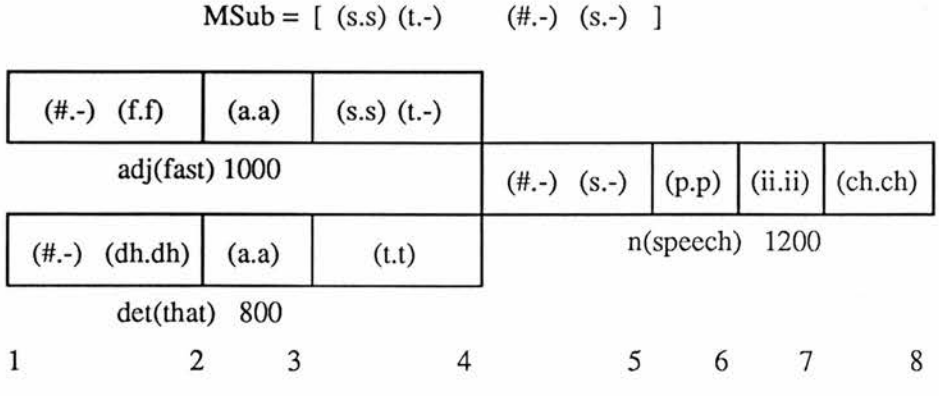


Figure 7–2: The alignment of the template 'speech' can be propagated from the alignments of either 'fast' or 'that'. The correct choice depends upon whether the MSub [(s.s)(t.-)(#.-)(s.-)] is applied.

However if the cross word boundary substitution [(s.s)(t.-)(#.-)(s.-)] is available this alignment is no longer optimal. A lower scoring alignment would have

been obtained by extending the alignment of 'speech' from the locally non-optimal template 'fast' since this allows the MSub to be applied. In this example the application of the positively correlated MSub outweighs the initial disadvantage of starting from the higher scoring alignment. The choice for the optimal template at the end of segment 4 depends upon whether the MSub is applied in the alignment of the following template. However, we can not know that the MSub will be applied until we have processed segment 6. In order to reach segment 6 we must first have calculated the optimal alignments to each of the preceding segments which will have originated from either 'fast' or 'that'.

In general it is not possible to correctly choose the locally optimal optimal templates until we know which if any word boundary spanning MSubs will be applied in the future. However we cannot calculate the alignments of future templates and determine whether MSubs should be applied until we have chosen the locally optimal templates from which the new alignments will be propagated.

Since it is not possible to know in advance what the correct choice of templates should be we construct a set of all templates that could be optimal depending upon whether MSubs were applied during the alignments of subsequent templates. From the previous discussion we have seen that it is possible for a locally non-optimal template to become part of the optimal alignment if a cross word boundary spanning MSub is applied between the alignment of the locally non-optimal template and the alignment of the following template.

The only locally non-optimal templates which are able to form part of the optimal alignment are those whose alignments could contain sequences of substitutions matching those which precede the internal word boundary deletion of a MSub. As we have seen a locally non-optimal template whose alignment could end with the sequence of substitutions (s.s)(t.-) may become part of the optimal alignment if the MSub [(s.s)(t.-)(#.-)(s.-)] is applied in the alignment of the following template. However, if the final sequence of substitutions could not correspond to any word boundary spanning MSub then the template could never be part of optimal alignment since there is no way that the alignment score could be decreased to offset the initial disadvantage of choosing this template.

The list of potentially optimal templates includes the locally optimal template and the lowest scoring templates whose final substitutions could correspond to the initial substitutions of a word boundary spanning MSub and which would score lower than the locally optimal alignment if the MSub was applied. Suppose we have four MSubs [(s.s)(t.-)(#.-)(s.-)], [(s.s)(#.-)(s.-)], [(s.s)(#.-)(p.b)] and [(z.-)(#.-)(s.s)] having correction factors 600, 300, 300 and 200 respectively. The set of potentially optimal templates, not including 'these', might be as follows.

Template	Reference Phonemes	Alignment Score
that	/dh a t/	1000
this	/dh i s/	1200
fast	/f a s t/	1500
these*	/dh ii z/	1800

The locally optimal template is 'that' which has the lowest alignment score. However the templates 'fast' and 'this' could enable a lower alignment score to be obtained for the following template by allowing the application of the MSub [(s.s)(t.-)(#.-)(s.-)], [(s.s)(#.-)(s.-)] or [(s.s)(#.-)(p.b)]. However, even though the inclusion of template 'these' could allow the MSub [(z.-)(#.-)(s.s)] to be applied the application of this MSub would still not produce a lower alignment score than would have been obtained by extending paths from the template 'that'. This template will never be part of the optimal alignment even if the MSub is eventually applied and therefore does not needed to be included in the set.

A decision has to be made about which of the potentially optimal templates the parser is going to be told about. The parser needs the information about the locally optimal templates so that it can determine the (hopefully small) subset of categories that will be valid in the next time interval. There would seem to be two obvious options. The first would be to inform the parser about all of the potentially optimal templates such as 'that', 'this' and 'fast' whilst the alternative would be to only tell the parser about the locally optimal template which in this case is just 'that' (this is what would happen by default if MSubs were not being used) and then take the necessary steps if this choice proves to be wrong. Neither

of these solutions is ideal, in the first case the parser will waste a large amount of time and storage in processing words which will never form part of the optimal alignment whilst in the second case it is possible that the parser will not have been told about words that should form part of the optional alignment.

We decided to chose the second option since the use of a chart parser allows the problem of missing templates to be easily overcome. The chart parser is initially told only about the locally optimal template in each category which is sufficient for it to be able to determine the set of valid categories for the next time interval. As the alignment of subsequent templates progresses it may turn out that a cross word boundary MSub is applied (how this actually occurs is described in the following sections) and that the template which is needed to provide the sequence of phoneme substitutions preceding the word boundary deletion was not in the list of templates that the parser was told about. The chart parser will have to be informed of the new choice for the preceding template so that the alignments within the chart and DP algorithm remain consistent.

Although the chart parser cannot make any changes to existing words once they have been added to the chart we can go back and add new alternatives to the chart. These new additions will cause edges to be scheduled and propagated in the usual way as described in chapter 5. The new lexical edge and any active or inactive edges which incorporate it will have a lower score than those already existing in the chart. From now on the DP algorithm used by the chart parser will propagate the new edges in preference to edges containing the now sub-optimal 'optimal' template. There is therefore no need to remove or modify any of the original edges.

7.2.2 A Solution

In anticipation of the problem with word boundary spanning MSubs we construct another tree based upon the original Dmin-tree produced by the tree decomposition algorithm described in the previous chapter. This tree has as its root a node corresponding to the word boundary deletion (#.-) and contains branches

for all sequences of phonemes which may precede an internal word boundary in a macro substitutions. Like all other trees that are processed during run-time the branches represent sequences stored in reverse order, i.e. successively deeper nodes correspond to progressively earlier substitutions. For example, two MSubs [(s.s)(t.-)(#.-)(s.-)] and [(s.s)(t.-)(#.-)(t.t)] contain the same initial sequence of phoneme substitutions (s.s)(t.-) which occur before the internal word boundary deletion and would therefore be represented by the single branch (#.-)(s.s)(t.-) in the pre-wb tree.

Each of the nodes in the pre-wb tree contains a score which is equal to the maximum score of all of the MSubs which have that particular sequence of substitutions preceding the word boundary. The score of the branch (#.-)(s.s)(t.-) would be equal to the highest correction factor associated with the two MSubs (s.s)(t.-)(#.-)(s.-) and (s.s)(t.-)(#.-)(t.t) along with any other MSub that has the same sequence of phoneme substitutions preceding the internal word boundary deletion. Each of the nodes in the pre-wb tree also contains an integer tag which uniquely identifies each node in the tree. The value of tag is 0 in the case of the root node and for the other nodes records the order in which they are visited during a depth-first descent. A fragment of a typical pre-wb tree is shown below.

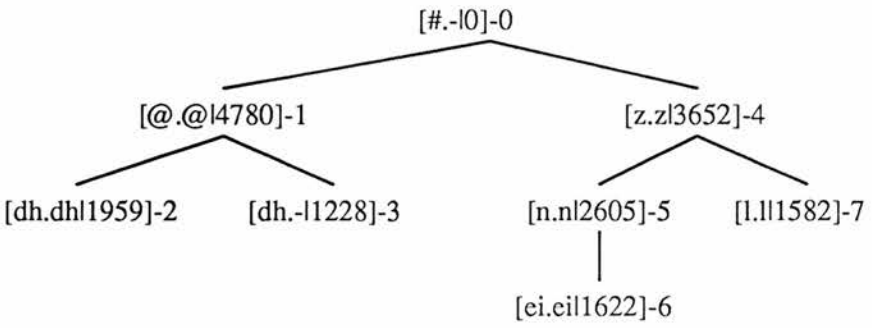


Figure 7–3: Small fragment from a pre-wb tree

The pre-wb tree is initially pruned against recent input segments by the pre-match pruning algorithm. This removes all redundant and useless branches from the tree which can not be matched with any of the substitution sequences in the current alignments. Once the DP processing has finished for the current time

interval, rather than simply calculating a single list of the lowest scoring templates, we attach a list of lowest scoring templates to each node of the tree. Each list has the same format as that which was used before by the standard DP algorithm and contains a list of the locally optimal templates, one for each category. Each node of the pre-wb tree contains a list of the lowest scoring templates subject to the constraint that their alignments could terminate with the sequence of phoneme substitutions depicted along the branch from the root to that node. A simplified example is shown in figure 7–4 which shows a list containing a single template attached to each node.

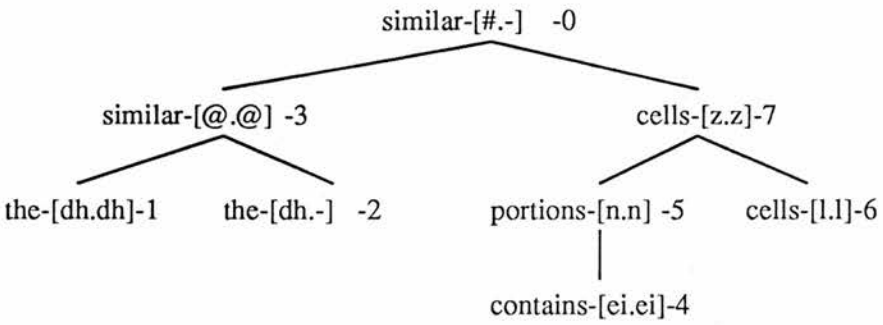


Figure 7–4: A pre-wb tree fragment containing the single best template subject to the phoneme constraints along the branches

Although the single root word boundary deletion node does not represent any substitution sequence it does contain the list of lowest scoring templates irrespective of their potential alignments. This is exactly the same list of the templates that was previously used in the standard context free DP algorithm. The other nodes in the tree contain the lowest scoring sequence of templates which could have alignments matching the substitutions on the nodes of the branch. For example in fig.7–4 node 4 would contain the list of best scoring templates which could end with the substitution (z.z), node 5 the list of best scoring templates ending which could end with the sequence of substitutions (n.n)(z.z) and so on with corresponding lists being stored on each node of the tree.

The tree of potentially optimal templates is constructed as follows. We start by matching the nodes of the tree, starting from the root and the last frame of

each template and work down depth-first through the tree as far as possible in order to determine the longest branch which is compatible with the corresponding phonemes in the template reference frames (we already know that the input phonemes are compatible since incompatible branches would have been removed from the pruned tree). Once the longest branch has been identified the alignment score of the template is compared with the scores of the templates already present in the list attached to the node. If the score is sufficiently low the template is inserted into the list. The alignment score of this template is then compared in the same way with the lists on each of the other nodes of the branch as we work back towards the root. This is necessary since the lowest scoring template whose alignment could end with the sequence of substitutions (ei.ei)(n.n)(z.z) may also be the lowest scoring template whose alignment could end with the substitutions (n.n)(z.z) or (z.z). The template score is also compared with templates stored on the root node (#.-) so as to produce the list of genuinely optimal templates irrespective of the final phoneme substitutions. This process of matching and comparing is repeated for all templates within each of the syntactic categories. At the end of this process the tree will contain the desired lists and the root node will contain the original list of optimal templates.

7.3 Pre-match Pruning

Pre-match pruning can achieve significant savings by virtue of the fact that each substitution (X.Y) in an MSub contains specifications for the input component 'Y' of each substitution in addition to the template component 'X'. It is therefore possible to rule out many of the potential P-MSubs and N-MSubs in advance. If the input component of any of the phoneme substitutions in an MSub are not consistent with the previously observed segments of the input lattice it will be impossible to find the corresponding path fragment in the DP matrix and also be impossible for it to be inserted as an additional path since the input phonemes required to perform the substitutions in the MSub do not exist. For example, in fig. 7–6 the sequence of phoneme substitutions (l.l)(-.l)(z.z) could not be present

at the end of any of the current template alignments nor could it be created since the required input phonemes /l l z/ are not present in the most recent segments of the lattice. It will therefore never be possible to successfully identify and apply MSubs which represent such paths so it is worthwhile to spend a little effort to eliminate them from further consideration.

The aim of pre-match pruning is to construct a pruned tree which contains the sub-set of branches from the original tree which correspond to potential path fragments. Nothing is lost by pruning these branches from the tree since, even if the branches were present in the tree they would never be used since the sequence which they represent could not occur in the DP matrix. The pre-match pruning algorithm is outlined below.

```
preprune(root, pruned_root, s)
  for c = 1 ...C(root)
    (X.Y) = symbol(c)
    if Y ∈ s
      C(pruned_root) = C(pruned_root) + (X.Y)
      if Y ≠ /-/ ps = s - 1
      if Y = /-/ ps = s
      preprune(c, C(pruned_root), ps)
```

Figure 7–5: The Pre-match Pruning Algorithm

The first step is to create a new root node `pruned_root` from which the pruned tree is grown. The original tree which may contain either P-MSubs, N-MSubs or pre-wb MSub fragments is then recursively descended depth first. The input component Y of the phoneme substitution on each of the $c = 1 \ldots C$ child nodes is matched against the input phonemes within segment s of the phoneme lattice, starting with the most recent segment and working backwards. If the input component is contained within the lattice segment the substitution (X.Y) is duplicated and appended to the children of the pruned root. The input phoneme

which matches the input component is also duplicated and inserted into a lattice sub-segment which is attached to the duplicated node. The process is repeated using the sub-trees that have the matched nodes as their roots and the preceding segment *ps* in the input lattice. If no corresponding input phoneme can be found or a childless leaf node is encountered the next sibling is processed. When all of the siblings have been processed the recursive loop exits and processing continues with the remaining siblings of the parent node.

The segment *ps* which precedes the input phoneme is located in the lattice at the starting time of the input phoneme. When the lattices are well structured, as is the case with our data, each of the phonemes in a segment have the same duration and in this case the preceding segment *ps* is usually simply equal to $s-1$. Although the AFE does not produce segments which explicitly contain the null phoneme /-/ the outcome of the test $Y \in s$ in the prepruning algorithm is always successful when Y is the null phoneme. Null phonemes have zero duration and can therefore be considered as always being present in an input segment irrespective of its actual contents. Since the duration is zero the segment which precedes these phonemes is also the segment which contains it i.e. $ps = s$.

During the initial stages of the DP alignment process many of the branches in the tree will have more nodes than there are segments in the lattice. Eventually, as the tree is descended the start of the input lattice will be reached and there will be no preceding segments whose contents can be compared with the remaining nodes of the tree. When this situation arises only those nodes having input components equal to the 'null phoneme' can be successfully matched since the success of this match does not depend on the contents of the (non-existent) segment. All other matches will fail since there is no segments or phonemes which can be matched with the input component.

The diagram in figure 7–6 shows the pruned trees which would be produced by pruning the initial tree against an input lattice containing just two segments. The pruned tree contains the sub-set of branches whose input components are consistent with the contents of the input lattice. This tree would be the output of the pre-match pruning procedure described above. The number of leaf nodes on the

pruned tree can be reduced further by removing leaf nodes that are insufficiently correlated. These leaf nodes can arise when matching fails at an internal node of the original tree which represents an insignificantly correlated sub-sequence, even though all leaf nodes in the original tree represent significantly correlated sequences.

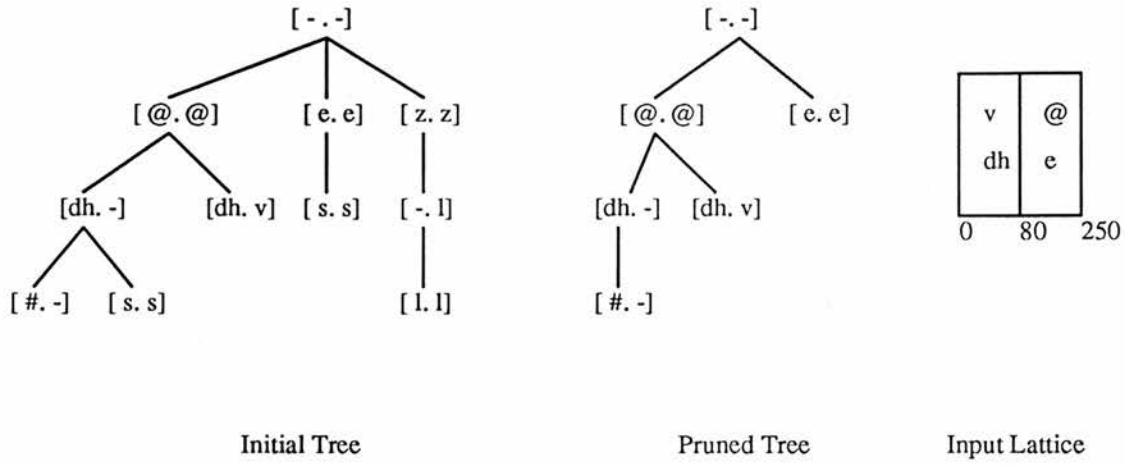


Figure 7–6: Pruning the initial tree against the input lattice produces the pruned tree which only contains those branches that are consistent with the contents of the lattice

The mechanically constructed trees produced by Treelab are always fully specified (i.e. they contain no broad class phonemes or prolog type variables) so pre-match pruning will always results in significantly smaller trees. However, nodes whose input component is the null phoneme /-/ will never be pruned from the tree unless one of their ancestors is pruned. For example, a branch which represents the sequence (dh.-)(@.-) only contains null input components and will never be pruned from the tree, even when the input lattice is empty! Fortunately, sequences such as these are extremely rare and in practice pre-match pruning always produces substantially smaller trees which are passed on to the subsequent stages.

7.4 Positively Correlated Macro Substitutions (P-MSubs)

The successful application of a positively correlated P-MSub will cause a new path fragment to be hypothesised with a reduced alignment score which may or may not already exist in the DP matrix. We already know that the input components of the phoneme substitutions in the MSub are consistent with the most recent lattice segments since these were checked when the pruned P-MSub tree was generated. Therefore we only have to check that the template components of the substitutions are consistent with the phonemes in the template reference frames in order to determine whether the path fragment corresponding to the P-MSub could have occurred in the alignment and should therefore be rehypothesised with a corrected alignment score.

Once a valid P-MSub has been identified the corrected score for the new path fragment is calculated and compared with the scores of any alternative paths in the DP matrix. If the score of the P-MSub is lower than that of any alternative path fragment the new path fragment is inserted into the DP matrix. The successful identification and application of a P-MSub occurs over three distinct stages. The first stage (matching) identifies applicable P-MSubs in the pruned tree, the second stage (scoring) calculates the context sensitive DP score of each applicable P-MSub, and the final stage (application) compares the context sensitive path score with the scores of alternative paths and inserts the path fragment into the DP-matrix if the score is sufficiently low.

Figure 7–7 will be used to illustrate the various stages which occur during the identification and application of P-MSubs. It should be clear from this figure that the input lattice segments (for simplicity each segment contains just a single input phoneme) are compatible with the input components of the P-MSub [(-.l)(-.l)(-.l)(z.z)] which requires a /z/ phoneme preceded by three /l/ phonemes to be present in the lattice. This P-MSub would therefore be present in the pruned tree at this stage in the DP alignment process.

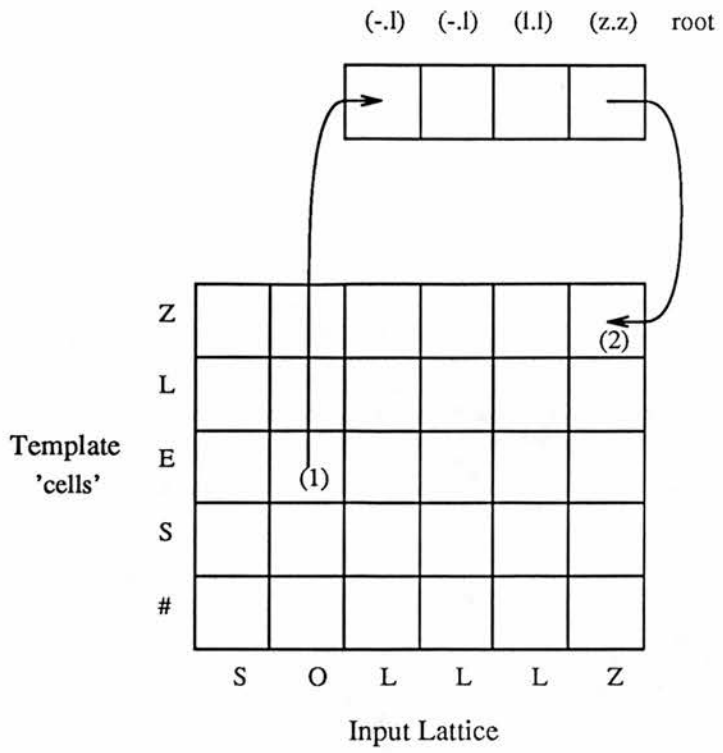


Figure 7–7: Positive Macro-Substitutions

7.4.1 Matching

The first stage identifies applicable P-MSubs in the pruned tree by matching the template component 'X' of each substitution (X.Y) with the corresponding reference frames of each template in the lexicon. At the end of each time interval the branches in the pruned P-MSub tree are matched against the reference frames j of each template k in the C categories. This is a computationally intensive process since the number of times the `match` procedure has to be called is proportional to the product of the number of templates in the lexicon and the average number of phonemes in each template. This is the penalty that has to be paid for applying phonological rules at run-time. If the equivalent phonological rules had been used to pre-compile the lexicon this stage would not have been required since the points at which rules were applicable would have been determined off-line during

the pre-compilation of lexicon. The points at which phonological rules are applicable would then be represented by alternative arcs in the network for the modified pronunciations. This is information is not available when phonological rules are applied at run-time so the DP algorithm must perform this computationally intensive matching stage at each time interval.

```
for c = 1...C
  for k = 1...K(c)
    for j = 1...J(k)
      match(P-MSubs,c,k,j)
```

The `match` algorithm which is outlined below is similar to the algorithm which was used for pre-match pruning. The P-MSub tree is descended depth first starting at the root node and the jth reference frame. The template component 'X' of the the phoneme substitution stored on each of the $n = 1 \ldots C$ child nodes is matched against the template reference frames starting at the reference frame j and working backwards towards $j = 0$.

```
match(root,c,k,j)
for n = 1 ...C(root1)
  (X.Y) = symbol(n)
  if X = phoneme(j)
    if X ≠ /-/ pj = pj - 1
    if X = /-/ pj = j
    match(n,c,k,pj)
  else apply(n,c,k,pj)
```

Figure 7–8: The P-MSub Matching Algorithm

If the template component and reference frame are compatible the matching procedure is repeated using the sub-tree that has the matched node as its root and the preceding template frame. This process is repeated until either a leaf node is reached or no successful match can be found. One consequence of the recursive

depth first search is that the most specific (i.e. longest) MSub will always be chosen in preference to other possible shorter sub-sequences.

Internal word boundary nodes (i.e. non leaf nodes) require special treatment. When a word boundary deletion has been successfully matched with the first ($j = 1$) frame of a template a special routine is called to match the remaining branches of the sub-tree which will have the word boundary deletion (#.-) as its root. The nodes in this sub-tree correspond to substitutions which precede the word internal deletion of an MSub. The template components of these substitutions will refer to the references frames of the preceding template in the alignment. In the preceding section we saw how a pre-wb tree was constructed which contains lists of all potentially optimal templates. This tree contains branches for all phoneme substitution sequences which can precede internal word boundary deletions in MSubs. Therefore the remaining branches in the sub-tree which also describe sequences of phoneme substitutions that precede the internal word boundary deletion will have corresponding branches in the pre-wb tree.

A similar recursive depth first matching procedure to that described above is used to find the branches in pre-wb tree which match the remaining branches in the sub-tree. The node in the pre-wb tree which is matched with a leaf node of the sub-tree will contain the list of templates that would have the lowest alignment scores if their alignments ended with the sequence of phoneme substitutions stored on each node of the branch. The cross word boundary spanning MSub is applied in turn using each of the templates in this list which can, according to the grammatical constraints, precede the current template.

Figure 7–7 illustrates the basic concepts of the matching procedure. The template components of the branch will be repeatedly matched depth first starting at the root and working backwards from each of the template's reference frames. This branch will only match the reference frames of the template 'cells' when matching starts at the last ($j = 5$) frame which contains the /z/ phoneme. The template components of this MSub require a /z/ phoneme to be preceded by a /l/ phoneme and two null phonemes /-/. The two null template components, like null input components, have zero duration and can effectively match nothing. Only the last

two reference frames have to match template components even of the MSub though the MSub contains four substitutions. The input and template components of the MSub are compatible with the contents of the most recent lattice segments and reference frames when matching starts at frame $j = 5$. It is therefore possible that the path fragment corresponding to this MSub will have been displaced from the DP matrix due to the high score which would have been calculated for it by the DP alignment algorithm.

7.4.2 Scoring

Whenever an applicable P-MSub is found the corrected DP score of the corresponding path fragment is calculated. This score is equal to the score of the alignment which will precede the substitutions in the P-MSub plus the scores of the individual substitutions in the MSub and the correction factor of the P-MSub. The first step is to locate the cell in the DP matrix which contains the accumulated distance score of the partial alignment which will precede the substitutions in the P-MSub (which is labelled (1) in fig. 7–7). This is the cell that would be reached by backtracking through the P-MSub if the P-MSub were present in the DP matrix.

The vertical slice through the DP matrix which contains the cell is located at the starting time $t_s(i)$ of the input phoneme i in the sub-segment s attached to the last node N to have been successfully matched. The cell within this slice is obtained from the index pj of the preceding template frame that would have been matched had the matching process continued. The partial alignment is then propagated from this cell on to the deepest node of the branch containing the matched P-MSub. In figure 7–7 the slice through the DP matrix is located at the lattice segment containing the phoneme /O/ since this segment precedes the last input phoneme that was successfully matched. Likewise, the cell in this slice is located at the template reference frame containing the phoneme /e/ since this frame also precedes the last template reference frame to have been successfully matched.

This cell contains information about the alignment which would precede the P-MSub. This information is used to extend the partial alignment onto the branch containing the P-MSub. The scores and backtracking pointers can then be calculated for the remaining nodes along the branch. At this stage we must distinguish between the accumulated distance scores which are stored on the nodes of the branch D_n and those which are stored in the DP matrix D_m. The accumulated distance score of the first substitution in the MSub which is stored on the deepest depth N node is calculated from the accumulated distance score of the preceding cell in the DP matrix as shown below. The backtracking pointers on this node will refer to the cell in the DP matrix.

$$\texttt{for } i = 1 \ldots I(s)$$
$$D_n(N, s) = D_m(c, k, s, j) + d(i, j) + ac(i)$$

The accumulated distance scores of the remaining substitutions in the P-MSub are calculated by working backwards from the deepest node ($n = N$) towards the root ($n = 0$), i.e. in the usual direction of increasing time. The iterative calculation of the path scores shown below is virtually identical to that which is used by the standard DP algorithm described in chapter 3.5.1. However in this case the paths are propagated between adjacent nodes in the branch instead of being propagated between cells in the DP matrix.

$$\texttt{Path Constraints}$$
$$D_n^*(n) = D_n(n + 1, t_s(i)) + d(i, P_n) + ac(i)$$

$$\texttt{for } n = (N - 1) \ldots 1$$
$$\texttt{for } i = 1 \ldots I(S_n)$$
$$D_n(n, t_e(i)) = \min D_n^*(n)$$

The path constraints are uniquely specified by the substitution stored on the current node of the branch. The only valid substitution is that which corresponds

to the substitution stored on the node of the P-MSub. For example, if the current node of the P-MSub contains the substitution (-.l) then this is the substitution which must be performed when calculating the score of the alignment of the branch, there are no alternatives. The scoring algorithm is outlined below and is same irrespective of whether the P-MSub contains an internal word boundary deletion.

The accumulated distance scores and backtracking pointers are calculated in the usual way using the template component component and the input phonemes stored in the sub-segments that were attached to each node of the branch during pre-match pruning. These substitutions are propagated from the previous node of the branch and consequently the backtracking pointers of a substitution will refer to the substitution stored on the preceding node rather than any of the paths in the DP matrix. The final score is obtained by subtracting the correction factor C^* which was stored on the deepest node ($n = N$) of the branch. The scored branch is then passed onto the routine described in the following section which compares the scores with existing paths and if necessary transfers the P-MSub from the branch into the DP matrix.

The DP calculations which are used to propagate the paths along the branch are much quicker than those which are used by the standard DP algorithm to propagate paths in the DP matrix. The template and input specifications on each node of the branch completely define the substitution which is to be taken to propagate the path. Consequently the path constraints consist of a single substitution (which may involve substituting with the null phoneme) rather than the three possibilities of substituting or deleting template and input phonemes which are used by the standard DP algorithm. The input component of each substitution are at present fully specified so each sub-segment contains a single phoneme compared with an average of six or more in the original lattice segments. This also means that fewer substitutions have to considered since each sub-segment contains fewer possible phonemes than the original lattice segments

Branch Score Caching

In the current version of Treelab it is not possible to extract MSubs which have substitutions containing broad class phonemes or PROLOG type variables. Broad class phonemes such as voiced plosive, front vowel or PROLOG type variables would enable template and input components to be matched with a range of different phonemes. Although this may allow much phonological knowledge to be represented more concisely in smaller MSub trees it is unlikely that such an approach would actually decrease the computational requirements of the matching, scoring, and application procedures for the following reasons.

First of all there would be less scope for pre-match pruning since the likelihood of a broad class phoneme or variable being compatible with the contents of a lattice segment would be much higher. So even though the initial trees may be much smaller the pruned trees are likely to be of a similar size to those which are currently obtained using fully specified phonemes. The use of broad class phonemes or variables would also rule out branch score caching which could also provide significant reductions in the computational requirements.

The optimal alignment of a path fragment represented by an MSub is uniquely identified by the sequence of substitutions which form the P-MSub. As we have seen in the previous section the corrected DP score of an alignment containing a P-MSub is equal to the score of the alignment which precedes the P-MSub plus the score of the individual substitutions of the P-MSub and its correction factor. Since the individual substitutions are fully specified the score of a P-MSub does not depend upon the phonemes in the template which were matched or the scores of any paths in the DP matrix of the template. This would not be true if any of the substitutions contained underspecified (broad class or variable) phonemes.

Suppose for example that a P-MSub contained the substitution (VP.UP) where VP and UP could be any voiced plosive (i.e. b,d,g) or unvoiced plosive (i.e. p,t,k). In this situation the penalties of the individual substitutions are no longer constant. The penalty of a substitution (b.p) would probably be much lower than the substitution (g.p). The alignment scores of P-MSubs such as these would depend

upon the phonemes which were matched with the template and input components. These can only be fully determined during the second P-MSub specific matching stage so it will be necessary to recalculate the alignment scores after every successful match. However, if all of the substitutions are fully specified like (b.p) and (g.p) the scores of each individual substitution and therefore the score of the entire MSub does not depend upon data from the template which is currently being matched. This is true because all /b/ and /g/ phonemes are alike irrespective of any template in which they occur.

It is therefore possible to precalculate (i.e. cache) the scores of P-MSubs before they are matched against the template reference frames rather than recalculating them after every successful match as has been described in the previous section and actually implemented. The alignment scores of the P-MSubs could be calculated once after the pruned tree had been generated and the lattice phonemes matching each input component had been determined. The subsequent alignment score of any template whose reference frames have been successfully matched is then simply equal to the score of the alignment which precedes the P-MSub plus the pre-calculated score of the P-MSub. The subsequent processing from this point onwards would then be identical for both techniques.

Although this technique has not been implemented, due to time constraints and the fact that the final alignments would be identical, empirical results presented in the next chapter show that this technique would produce significant reductions in the computational overheads that are introduced by the incorporation of MSubs.

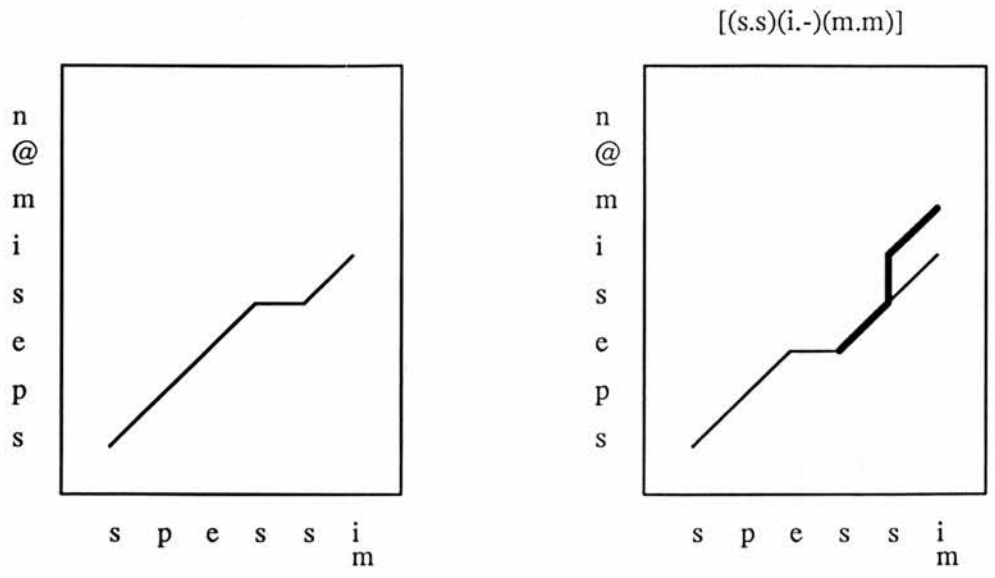
7.4.3 Application

The corrected alignment score of a P-MSub is compared with the scores of alternative paths in the corresponding cell of the DP matrix (labelled 2 in fig. 7–7). If the path score is sufficiently low the path fragment is transferred from the branch and into the matrix where it can be extended along with the other paths during subsequent stages of the DP alignment process. Data from the first node of the P-MSub (i.e. (z.z) in fig. 7–7) can be inserted directly into the corresponding

cell in the DP matrix and will replace the old accumulated distance score and backtracking pointers with new values. Since this simply involves replacing one set of data with another *there is no increase in the amount of data that has to be stored.* This single step would be sufficient if only template level backtracking was required and we were not interested in backtracking through the actual sequence of phoneme substitutions that make up the optimal alignment.

When phoneme level backtracking is required the accumulated distance scores and backtracking pointers will also have to be transferred from each internal P-MSub node in to the corresponding DP matrix cell. However, these can not be used to simply replace existing paths as was the case with the first node of the P-MSub. A P-MSub which is transferred into the DP matrix should be treated as an atomic operations like any other substitution and it should not be possible to splice one P-MSub into the middle of another. Doing so would almost certainly lead to incorrect results.

Consider for example the two diagrams in figure 7–9. The first diagram 7–9(a) shows one optimal partial alignment (s.s)(p.p)(e.e)(s.s)(-.s)(i.i) of the template 'specimen'. The application of the P-MSub [(s.s)(i.-)(m.m)] produces a new optimal alignment (s.s)(p.p)(e.e)[(s.s)(i.-)(m.m)] shown in diagram (b). If the accumulated distance scores and backtracking pointers from the internal nodes were simply copied straight back into the DP matrix as indicated by the bold black line the backtracking pointers for the original path would be overwritten. Although the final cell and score of the original alignment has not changed the path which would be obtained by backtracking from this cell has. This path is now sub-optimal and the new sequence of substitutions is not consistent with the alignment score stored in the cell. Clearly P-MSubs should be treated as atomic operations and it should not be possible for the internal nodes of a P-MSub to alter the original context free DP paths or paths produced by the previous application of P-MSubs. We therefore need to be able to distinguish between the last node of the P-MSub which can be extended by the DP algorithm and the internal P-MSub nodes which should only be used for backtracking purposes and kept separate from those in the DP matrix.



(a) Optimal partial alignment before the application of P-MSubs

(b) Optimal partial alignment after the application of P-MSubs

Figure 7–9: Two partial alignments of the template 'specimen', one before and one after the application of the P-MSub [(s.s)(i.-)(m.m)]

These problems are overcome by expanding each cell in the DP matrix. Each cell now has a list which contains context sensitive path data from the internal nodes of the P-MSub in addition to the original list which contain data about the context free paths and P-MSub applications. The standard DP algorithm only looks at paths on the context free list when propagating new paths. The final node of the P-MSub is inserted into this list where it can be extended along with the other paths in the matrix. The remaining nodes of the P-MSub are transferred on to the context sensitive list. The contents of the context sensitive list are ignored during subsequent stages of the dynamic programming alignment process and are only accessed once the alignment is complete and the sequence of phoneme substitutions is to be displayed. The backtracking information of paths in each list needs to be extended so that each substitution now contains a flag which indicates whether data about the preceding substitution is stored on the context free or context sensitive list.

When the branch in fig. 7–7 is transferred into the DP matrix the flag stored on the context free list in cell (2) will redirect the backtracking onto the context sensitive lists which will cause the internal nodes of the P-MSub to displayed. Then, when cell (1) is reached backtracking will be redirected back on to the context free lists and the remaining DP substitutions will be displayed as usual. One useful consequence of separating context free and internal P-MSub substitutions is that it permits portions of the alignments that were produced by the application of P-MSubs to be easily identified and displayed as shown in app. A.5.

7.5 Negatively Correlated Macro Substitutions (N-MSubs)

Negatively correlated macro-substitutions (N-MSubs) describe sequences of phoneme substitutions that should have a higher alignment score (i.e. lower probability) than would have been expected from the context free substitution penalties. The standard context free dynamic programming algorithm will therefore underestimate the alignment scores of these path fragments. The alignments which make use of these path fragments will have artificially low scores and, since the dynamic programming algorithm always choses the lowest scoring paths, they may displace the genuinely optimal path from the DP matrix. This could lead to errors in the final alignment along with a decrease in the overall recognition accuracy.

N-MSubs are therefore used to prevent the creation of paths with artificially low scores. In order to achieve this we must check at the start of each stage of the DP alignment process whether any substitution would lead to the creation of a path fragment corresponding to a N-MSub. When such a substitution is found the N-MSub correction factor C^* is added to the substitution penalty before the new path is compared, as before, with the other paths in the DP matrix

This process also takes place in three distinct stages. In the first stage pre-match pruning removes redundant branches from the N-MSub tree. Then during the second stage the paths in the DP matrix are compared with the remaining

nodes in the tree in order to identify those substitutions which would lead to the creation of negatively correlated path fragments. The final stage involves monitoring the DP algorithm so that when these substitutions are considered the appropriate correction factor is incorporated into the substitution penalty calculations. The last two stages are specific to the application of N-MSubs and these are described in the following sections of this chapter.

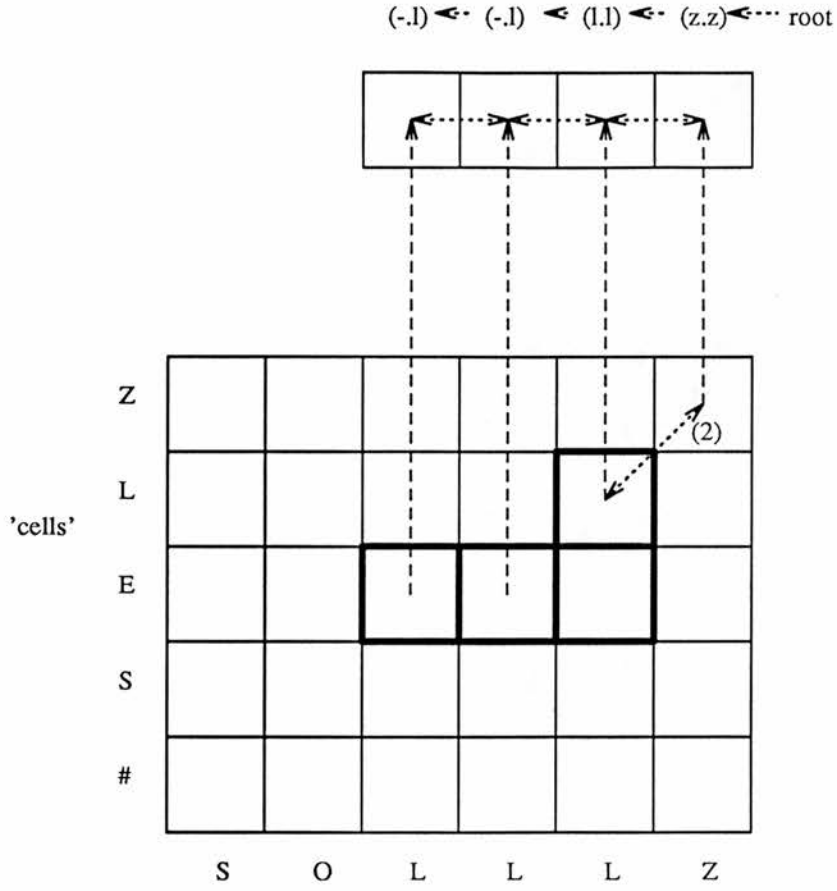


Figure 7–10: Negative Macro-Substitutions

Figure 7–10 will be used to illustrate the various stages which would occur during the application of the N-MSub [(-.l)(-.l)(l.l)(z.z)]. Although this sequence is actually a P-MSub it is useful to use this since it highlights the differences between the application of N-MSubs and P-MSubs described in the previous section. As before it should be clear that the input components of each substitution in N-MSub are compatible with the contents of the most recent input lattice segments. The N-MSub requires a /z/ phoneme preceded by three /l/ phonemes which can

be seen to present in the lattice. This N-MSub would therefore be present in the pruned tree at this stage of the implementation.

7.5.1 Matching

The first stage determines which of the N-MSubs in the pruned tree are applicable. In order to achieve this we compare the phoneme substitutions on the nodes of the tree with existing substitutions in the DP matrix. We have have to check that the substitutions actually are present rather than simply checking that they could be (have been) present in order to be able to identify those substitutions which would lead to the creation a path fragment corresponding to an N-MSub. For example, to ensure that the path fragment corresponding an N-MSub [(-.l)(-.l)(l.l)(z.z)] is scored correctly we would first look in the DP matrix for the sequence of substitutions (-.l)(-.l)(l.l) in the DP matrix. If this sequence is found we monitor the DP algorithm and incorporate the N-MSub correction factor in to the substitution penalty calculations when the substitution (z.z) occurs. This ensures that the correct alignment score for the substitution sequence (-.l)(-.l)(l.l)(z.z) is calculated before it has a chance to (incorrectly) displace other paths in the DP matrix.

The matching procedure is more complex than that which is used to identify P-MSubs since we must also check that the path fragments are actually present in the DP matrix rather than just checking that they could be present. At the start of each time interval, before the standard DP processing begins, the nodes of the pruned N-MSub tree are compared against existing paths in the DP matrix. The comparison begins with the depth two nodes (since the substitutions corresponding to the depth one nodes will not have occurred yet). These branches are matched depth-first matching the template component 'X' of each substitution (X.Y) with the corresponding reference frames of the template. Each of the $K(c)$ templates in the C categories are matched in turn with the matching process starting at each of the $J(k)$ reference frames j of template k. As was the case with P-MSubs this is a computationally intensive process since the number of times the `match`

procedure is called is also proportional to the product of the number of templates in the lexicon and the average number of reference frames in a template.

```
for c = 1...C
  for k = 1...K(c)
    for j = 1...J(k)
      match(N-MSubs,c,k,j,s)
```

Each descent of the N-MSub tree results in the comparison of progressively earlier phoneme substitutions. Each of these substitutions is found by following the phoneme level backtracking pointers which are stored in the cells in the DP matrix. In addition to checking that the phonemes in template component and template reference frame j are compatible we also have to check the backtracking pointers in order to ensure that the actual substitutions are in fact present in the DP matrix. One consequence of the recursive depth first search outlined below is that the most specific (i.e. longest) N-MSub will always be chosen in preference to other shorter N-MSubs.

```
match(root,c,k,j,s)
for n = 1 ...C(root1)
  (X.Y) = symbol(n)
  if X = phoneme(j) and substitution(j,s) = (X.Y)
    if X ≠ /-/ pj = pj - 1
    if X = /-/ pj = j
    if Y ≠ /-/ ps = ps - 1
    if Y = /-/ ps = s
    match(n,c,k,pj,ps)
  else apply(n,c,k,pj)
```

Figure 7–11: The N-MSub Matching Algorithm

The template component 'X' stored on node n of the pruned tree is matched against the phonemes in the references frame j, working backwards towards $j = 0$

and $s = 0$. The backtracking pointers of the cell (j,s) are also checked to ensure that the substitution is actually present in the DP matrix. If the template component and reference frame are compatible the matching procedure is repeated using the sub-tree that has the matched node n as its root, the preceding template frame pj and the preceding lattice segment ps. The preceding lattice segment is also determined since this along with the preceding reference frame identifies the cell in the DP matrix which should be matched against the children of the current node. This process is repeated until either a leaf node is reached or no successful match can be found.

Branches containing internal word boundary deletions would also require special treatment. In practice this would be similar to that which was required to handle internal word boundary deletions in P-MSubs. This would involve matching the substitutions of the remaining sub-tree with the substitutions in the alignment of the template which preceded the current template. However, since we do not have sufficient data to be able to reliably extract any N-MSubs which span word boundaries the necessary techniques have been neither implemented nor tested. We therefore omit details of the algorithm which would be required to handle this special case.

Figure 7–10 illustrates the basic concepts of the matching procedure. The template components of the branch will be repeatedly matched depth-first starting with the depth-2 node containing the substitution (l.l) and working backwards from each of the template's reference frames and the current lattice segment. As before this branch will only match the reference frames of the template 'cells' when matching starts at the last ($j = 5$) frame. Each of the N-MSub substitutions must have have corresponding cells in the DP matrix which contain the same substitutions. These cells have bold outlines in fig. 7–10. The substitutions or deletions which were performed can be determined by the phoneme level backtracking pointers which are indicated by the dotted lines. The N-MSub will be successfully matched when starts at frame $j = 5$. The sequence of substitutions (-.l)(-.l)(l.l) is therefore present in the DP matrix and the substitution (z.z) will, when performed by the DP algorithm, complete the N-MSub. In the next stage

we describe how the DP alignment algorithm is monitored so that when this substitution is performed the correct alignment score is calculated before this path fragment has a chance to incorrectly displace alternative paths.

7.5.2 Application

Whenever the matching procedure terminates at a leaf node or negatively correlated internal node additional care must be taken during the application of substitutions corresponding to the depth-1 node of the branch. Success at the matching stage indicates that a path fragment exists in the DP matrix which will match a N-MSub if the next substitution matches the depth one node of the branch. As we have seen, the standard DP algorithm if allowed to proceed unchecked will generate a path score which is artificially low. We must therefore intercept these substitutions as they occur and apply the appropriate correction factor so as to obtain the correct score before the path is considered for insertion into the DP matrix.

This is achieved by creating a list of guard cells which contains the substitutions stored on the depth one node of each successfully matched N-MSub (such as (z.z) in fig. 7–10). The correction factor C^* of this N-MSub is retrieved from the last node to have been successfully matched and is stored along with the substitution on the guard list. Before any of the substitutions are considered by the DP alignment algorithm they are compared with the substitutions stored in the list of guard cells. If this list is empty or no successful matches can be found the substitutions are performed as before and their scores are calculated in the usual manner. However, if a successful match is found the substitution penalty is increased by adding the correction factor which was stored in the guard cell. The corrected accumulated score is then compared with the accumulated distance scores of alternative paths in the DP matrix and from here on processing continues as before.

The list of guard cells is destroyed once all of the paths into the current cell (j,s) have been propagated. The entire matching and implementation processes are then repeated for each of the remaining template reference frames. Unlike the

application of P-MSubs no additional data or backtracking information needs to be preserved since no new paths have been created. This means that *there is no increase in the amount of data that has to be stored.* This also means that we have no way of showing where N-MSubs have been applied. It is difficult to show those path fragments that have been prevented from occurring in the DP matrix!

7.6 Summary

In this chapter we described how optimal DP alignments containing correlated sequences of phoneme substitutions (i.e. MSubs) can be obtained by adding an additional layer of processing to the standard context free DP alignment algorithm. The context free DP algorithm (Ch. 3.5.1) remains at the heart of the context sensitive DP alignment algorithm and propagates the partial alignments as usual under the assumption that the individual substitution penalties are independent. Correlated sequences of substitutions whose alignment score differs significantly from the sum of the individual substitution penalties will be present in the MSub tree which is constructed as described in the previous chapter. The branches in this tree describe sequences of phoneme substitutions whose alignment scores will have been incorrectly calculated by the context free DP algorithm. The context sensitive DP algorithm looks for these sequences and calculates the correct score for them as and when they occur.

The correlated phoneme substitutions sequences (MSubs) are split into two types depending upon whether the sequence is positively correlated (P-MSubs) or negatively correlated (N-MSubs). P-MSubs and N-MSubs have to be treated in different ways as these have different implications for the DP alignment algorithm. The original MSub tree is broken up into smaller sub-trees in order to facilitate the independent processing of P-MSubs, N-MSubs and MSubs which span word boundaries. Three smaller trees are derived which contain only P-MSubs, only N-MSubs and fragments of MSubs which precede internal word boundary deletions.

The latter pre-wb tree is required to implement MSubs which span word boundaries and thereby introduce context dependencies between sequences of templates.

The three different implementation classes are shown in figure 7–12. The first stage in each class is always pre-match pruning which takes place at the beginning of each time interval. The pre-match pruning algorithm recursively descends each tree depth-first comparing the input component of each substitution with successively earlier segments of the input lattice. MSubs whose substitutions would place template phonemes into correspondence with input phonemes which are not present in the appropriate segment of the lattice can not be produced by the DP alignment algorithm. It is impossible for these sequences to match any of those sequences currently being propagated by the context free DP algorithm so they are pruned from the tree. This preprocessing stage saves an enormous amount of work since the amount of computation required in the following stages is proportional to the product of the size of the (pruned) tree and the size of the lexicon.

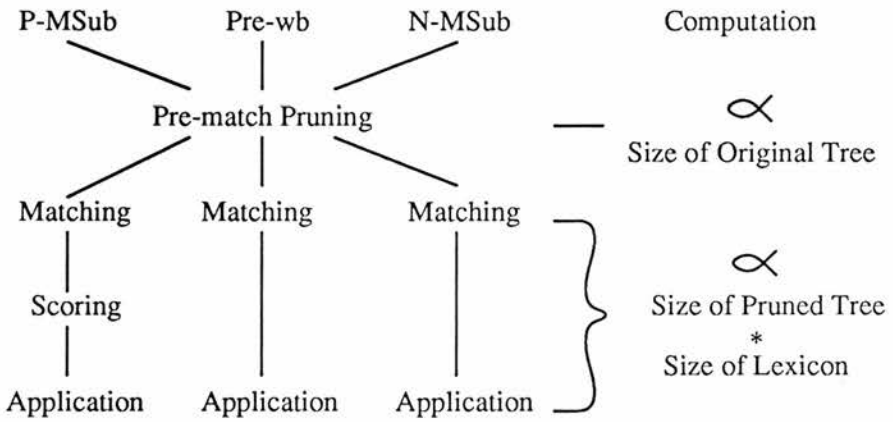


Figure 7–12: The three MSub implementation classes

The next matching, scoring, and application stages are specific to each class of MSub. The template components of the remaining substitutions in the pruned tree are matched with the reference frames of each template in the lexicon. This is a repetitive but simple procedure which differs alightly according to whether we wish to determine if a path could have been present in the DP matrix in the case of the P-MSub or pre-wb tree, or is definitely present in the DP matrix in the case

of the N-MSub tree. The scoring and application processes are more complex but these only applied to those MSubs which have been successfully matched. These processes also differ according to the class of MSub tree.

The amount of additional storage which is required when implementing P-MSubs or N-MSubs is either small when phoneme level backtracking is required or zero when only template level backtracking is required. In practice template level backtracking is usually sufficient since the sequence of templates, rather than the details of their alignments, is what is required from a speech recognition system. We also showed that provided the P-MSubs contained no broad class phonemes or PROLOG type variables it would be possible to effectively eliminate the additional scoring step for P-MSubs. This could be achieved by performing a one-off pre-calculation of the P-MSub alignment scores after pre-match pruning rather than after every successful match against each template in the lexicon. This is possible because the sequences of substitutions in the P-MSubs are well defined and the alignment score is independent of the preceding alignments. Quantitative results along with a discussion which compares this approach with alternative methods of incorporating phonological knowledges are presented in the next chapter.

Chapter 8

Results and Conclusion

8.1 Context-Free Dynamic Programming

We have shown in chapter 3 how the traditional and computationally efficient one-pass and level building DTW algorithms can be adapted for use in the naturally discrete problem domain of lexical access. This required a reformulation of the path constraints which in this domain control the insertion and deletion of discrete symbols rather than the compression and expansion of what were essentially continuous speech vectors. The use of discrete symbols led to problems with phrase and word initial boundaries. The DTW between-word path constraints do not allow word or phrase initial deletions but these are often necessary for obtaining the optimal alignment of naturally discrete sequences. These problems were overcome by the the addition of a second stage of dynamic level building into the discrete one-pass DP algorithm which is used for lexical access. This additional stage allows phrase and word initial deletions to be used whilst preserving the relative advantages of the one-pass algorithm over other less efficient algorithms.

The addition of a constant word offset penalty [McKelvie 90] provided the most dramatic increase in word recognition scores. The score offset was added to the initial score of each new template hypothesis and had the effect of reducing the number of templates and hence fortuitous matches in the final alignments. This also had the unfortunate effect of eliminating most function words such as 'the' and 'of' which were usually aligned with a relatively small number of poorly recognised (i.e. high scoring) phonemes. However the increases in word recognition score which were obtained due to the elimination of fortuitous matches more

than outweighed the decreases due to missing function words. The addition of a constant word offset corresponding to approximately to half the average phoneme substitution penalty increased the word recognition rate from 31% to 60%.

8.1.1 Enhancements and Extensions

A corrective training procedure was proposed which attempts to reduce the number of significant differences between the frequencies of phoneme substitutions that were observed in the alignments produced by the training and evaluation algorithms operating on the same set of phoneme lattices. The corrective training algorithm increased the substitution penalty of those substitutions which occurred too often and decreased the penalties of those substitutions which did not occur often enough. This caused most of the missing function words to be retrieved but also led to the reintroduction of many fortuitous matches which required a further increase in the word offset penalty. Once completed corrective training produced a small but consistent increase in the recognition rates giving an average word recognition rate of 66%.

The discrete nature and relatively broad temporal quantisation of the input lattice meant that the duration and temporal location of a template alignment was usually well-defined and unambiguous. It was shown that when the starting and finishing times of the N-best alignments within a template are identical the optimal sequence of substitutions along each alignment are also identical and independent of the initial alignment score. This means that each of the alignments having the same starting and finishing times can be obtained from a single alignment that also has the same starting and finishing times. An analysis of the alignments obtained during training showed that it would have been possible to calculate over 99% of the 10-best alignments by calculating just the 2-best alignments within each template. This results in a saving of over 80% in storage and computational requirements compared a full implementation of the N-best algorithm with negligible losses in optimality.

We also investigated the extent to which beam width pruning could be used to further reduce the storage and computational requirements. We found that there was an optimal range of values for the beam width pruning threshold which, in our application, lay in the range 1200 to 1400. Using a beam width pruning threshold in this range produces a further reduction in the search space of around 50%. Using smaller beam width thresholds than 1200 was shown to result in the loss of many more of the genuine N-best paths without producing a corresponding increase in the reduction of the search space.

8.2 Automatically Extracting MSubs

Treelab was applied to the four different partitions of 60 training sentences from the original corpus containing 80 cytology utterances. The optimal alignments between the phoneme lattices and the correct sequence of lexical templates were then processed by treelab to produce MSub trees for each of the four corpus partitions. The set of 60 alignments from each partition yielded an average of 2500 phoneme substitutions from which the trees were constructed. The decomposed trees of positively correlated P-MSubs contained an average of 211 nodes, each of which correspond to a potential P-MSub. The pre-word boundary tree contained on average 37 nodes and the tree of negatively correlated Msubs contained an average of just 18 nodes. An example of typical P-MSub, N-MSub, and pre-wb trees are shown in the appendix (app. A.4).

Although word boundaries are a potentially rich source of phonological variation the number of reliably detected MSubs which span word boundaries, as indicated by the average size of the pre-wb tree, is comparatively small. Most of the sequences in the training alignments do not span word boundaries and those that do usually did not occur frequently enough to enable reliable estimates of the probabilities and correlation coefficients to be calculated. Consider for example, a well known phonological assimilation rule 's $\mapsto \phi$ / _ # s' which merges two /s/ phonemes into one when they occur on both sides of a word boundary. This

phonological rule is equivalent to the P-MSub [(s.s)(#.-)(s.-)]. This MSub was not present in any of the P-MSub trees since this sequence was only ever observed once in the entire set of 60 alignments. This is not often enough to be able to reliably determine whether the sequence is sufficiently correlated and consequently the branch would be pruned from the tree. Since this sequence, and others like it, occur so rarely in the correct alignments it is also unlikely that they will have much of a beneficial effect if they were included in P-MSub tree.

The N-MSub tree also contains very few nodes compared with the P-MSub tree since these require significantly more training data in order to be reliably detected. The a-priori probabilities of the individual phoneme substitutions are comparatively small and those of the negatively correlated sequences will be even lower. As the probabilities of the negatively correlated sequences are so low we can not expect to see many (often less than 1) occurrences of any particular sequence in the training alignments. Therefore it is extremely difficult to actually find those sequences which are occurring less often than would have been expected from the a-priori substitution probabilities. If, for example, the probability of a particular phoneme substitution was 5% (which is in fact very high), we would only expect to see a single occurrence of this substitution after 20 occurrences of any particular sequence of substitutions. However there are very few substitution sequences in the training corpus which occur 20 times. In practice many more observations would be required in order to reliably determine whether the substitution was occurring less often than expected and was therefore negatively correlated with the surrounding substitutions

Many more N-MSubs could have been generated if we had added final pass nodes [Rohwer$_{\wedge}^{\text{In Press}}$] when constructing the MSub trees. Final pass nodes are leaf nodes which have a zero tally (uncertain by one) and thereby allow sequences which were never seen in any of the alignments to be represented in the tree in the same way as the other nodes. Of course, there are an infinite number of sequences which were not observed and it is obviously impractical to consider representing all of them. It also seems unlikely that the branches which would be generated by the addition of zero-leaves would be useful in practice. Most of the branches

that would be created would be useless since there would often have been good reasons (e.g. phonotactic constraints) why the sequence of phoneme substitutions had not occurred in any of the training alignments. For example, it would be pointless to add a branch for the sequence (s.s)(s.-)(s.-) since there is no template in the lexicon which contains three consecutive /s/ phonemes. Even though the substitutions in this sequence are definitely negatively correlated and the sequence has a lower than expected probability it would be useless to try and use it as an N-MSub since it could never be successfully matched with the reference frames of any template.

Although more training data would certainly enable subtler phonological phenomena to be detected and more accurate correction factors to be calculated the amount of training data does not seem to be a major factor in limiting the recognition performance of the system at present. Even if we had a sufficiently broad corpus of training data to enable correlated substitution sequences arising from the application of phonological rules such as /s # s/ $\mapsto$ /s/ to be automatically detected and extracted it is unlikely that the corresponding MSubs would have a significant positive impact on the recognition performance of the system. If the sequence of substitutions in the P-MSub [(s.s)(#.-)(s.-)] occurs in less than 1% of the correct alignments there is little chance that its application will result in the correct alignment of an utterance. Of course, the P-MSubs could also be applied in many other situations due to fortuitous matches with template reference frames and lattice segments which may degrade the overall recognition performance.

The potential advantages which could arise from the incorporation of MSubs representing infrequent sequences are small but the negative consequences could be large since the number of fortuitous applications is likely to be much larger than the number of correct applications. In the example given the MSub could be applied whenever a lattice segment contained an /s/ phoneme. We have shown by hand constructing an optimal MSub tree (Ch. 8.4.3) that significant increases in recognition accuracy can be achieved by pruning most of the branches from the tree. The problem is therefore to determine which of the existing MSubs should be

discarded rather than using larger amounts of data to extract even more MSubs representing subtler phonological phenomena.

8.3 Interpreting MSub Trees

It was our original intention the extracted MSubs would account for the effects of human and machine phonology. However only a small proportion of the branches in the trees (e.g. app: A.4) can be readily identified as accounting the effects of either human phonology or machine phonology. The majority of the branches in trees appear to depict sequences that are artifacts of our limited training data and restrictive domain. Although we had not intended that these kinds of sequences would dominate the trees as we shall see they can still be just as useful as those sequences which do correspond to human and machine phonology.

8.3.1 Human and Machine Phonology

There are a few sequences in the MSub trees (app: A.4) that could represent the effects of human phonology such as (@.@)(v.-) (from 'of'), (s.s)(ɹ.dɦ)(m.m) (from 'smears'), and (e.e)(s.s)(i.-)(m.m) (from 'specimen') as each of these sequences describe plausible alternative pronunciations. However the sequences alone are not sufficient to enable a decision to be made about which sequences definitely resulted from human phonology and those which actually resulted from machine phonology. Although (@.@)(v.-) might be a valid pronunciation of 'of' in fast speech and could therefore be ascribed to human phonology it is also possible that what was actually uttered was /@ v/ but the AFE failed to detect the /v/ phoneme.

There are a few sequences which are more likely to have arisen due to machine phonology. We can find basic mis-recognitions such as (e.o), (p.g) and (k.t) as well as the mis-segmentation of some diphthongs and liquids such as (i@.i)(-.@) and liquids (-.l)(-.l)(l.l). Information such as this highlights deficiencies in the AFE

and shows where effort needs to be concentrated in order to increase the basic phoneme recognition accuracy of the acoustic front end.

It has been suggested [Picone *et al* 86] that the confusion matrices from which the a-priori substitution penalties were calculated could be used to identify areas of weakness within the acoustic front end and evaluate subsequent modifications. The confusion matrices (for example app. A.3) contain the tallies of each of the identity and non-identity substitutions (including insertions and deletions) that appeared in the optimal alignments of the training sentences. These have been found to be useful in identifying those phonemes that were poorly recognised (i.e. occurred in non-identity substitutions) or were commonly inserted or deleted. They have also been found to provide a reliable and consistent means of comparing the relative performances of different acoustic front ends.

However the confusion matrices provide no information about the context in which mis-recognitions, insertions, or deletions occur. Examination of the confusion matrices might for example show that it was fairly common for the schwa phoneme /@/ to be deleted which would not be unexpected. We might also notice that the phoneme /i/ was occasionally substituted with (or mis-recognised as) the diphthong /i@/. Neither of these non-identity substitutions would appear to indicate particularly serious weaknesses on their own. However, if we could look at the context in which these errors occurred we would see that a small fraction of the /@/ phoneme deletions could be accounted for by the fact that a input /@/ was always deleted after after a (i@.i) non-identity substitution due to the consistent mis-segmentation by the AFE of the diphthong /i@/. Weaknesses such as these cannot be identified solely from the confusion matrices.

The N-gram and MSub trees produced by Treelab provide the same information as the confusion matrices in the depth one nodes. However the trees also provide information about the context in which the (non-identity) substitutions commonly occurred. The tallies of the depth one nodes in the N-gram tree record the frequency of each substitution which is exactly the same information that is present in the confusion matrices. The longer branches of the tree record the tallies of longer sequences such as (i@.i)(-.@) and these can be used to identify

errors arising from mis-segmentations or context-sensitive recognitions. We therefore believe that N-gram and MSub trees provide a more useful representation for identifying potential problems which will enable a more detailed assessment to be made of the relative performances of different AFEs.

8.3.2 Domain Dependent Artifacts

The majority of the branches in the tree appear to represent domain dependent artifacts which arise because of the limited number and range of our training sentences. The sequence (dh.dh)(@.@)(#.-)(s.s) is usually present in the tree because the phrases 'the smears' and 'the cells' are particularly common in the sentences in our corpus. Therefore the probability of a (s.s) substitution is higher after the sequence of substitutions (dh.dh)(@.@)(#.-) and this results in the presence of the P-MSub [(dh.dh)(@.@)(#.-)(s.s)] in the tree. Other branches in the tree represent other common phrases or fragments of common words such as 'specimen' and 'contains'.

The application of P-MSubs such as these will tend to enforce a weak stochastic grammar. In the case of (dh.dh)(@.@)(#.-)(s.s) the probability of segmenting the sequence of input phonemes /dh @ s/ (the AFE does not recognise word boundary phonemes) into /dh @ # s/ by performing the substitutions (#.-)(s.s) after (dh.dh)(@.@) is increased. This also increases the probability that words such as 'cells' and 'some' will occur after alignments of the template 'the'. However, this also increases the probability of obtaining invalid phrases such as 'the some'. As we shall see, although there are cases where this effect does lead to the uncovering of the correct alignment there are also cases where this actually leads to poorer word and sentence level recognition scores.

Although the realisation of weak stochastic grammars is an interesting effect we do not wish to claim that this is a sensible or efficient way of enforcing grammatical constraints. Furthermore, as the grammatical complexity of the training corpus and lexicon size increase most of these artifacts will tend to fade away. It still seems likely though that useful information about which phonemes tend to occur

at the beginning or ends of words could still be extracted and incorporated into the alignment process. As the size of the corpus and lexicon increases we would expect to be able to extract information about how phonemes tend to cluster together to form syllables. At the present a few of the branches do consist of sequences of phoneme substitutions that can be identified as corresponding to syllables. Examples include (n.n)(y.y)(uu.uu) from 'nuclei' and 'neutrophils' and (y.y)(u@.uu)(r.r) from 'urine' and 'urothelial'. P-MSubs such as these could still be used to implement a kind of stochastic syllabic grammar for phonemes. Syllables may be useful as an intermediate level of representation to aid recognition and might also be used to provide information about phonological processes which depend upon the location of phonemes within syllables.

8.4 Run-time Implementation of MSubs

The table below shows the word and sentence recognition rates that were obtained from the DP algorithm using all possible combinations of parser and MSubs.

	Word Recognition Rate	Sentence Recognition Rate
No parser and no MSubs	58.7%	7.5%
Parser but no MSubs	59.8%	10.0%
No parser and Msubs	61.3%	10.0%
Parser and Msubs	62.3%	15.0%

These results were obtained by evaluating each of the possible combinations in turn on four different partitions of our cytology corpus. Each partition consisted of sixty sentences which were used to train the DP penalty matrix and construct the MSub tree and the remaining sentences were used to evaluate the DP algorithm. This process was repeated for each of the four partitions and for each combination of parser and MSubs. The results show that there is a slight improvement in the word recognition scores as we progress from using the DP algorithm without the parser and MSubs to the DP algorithm which uses both. The increase in sentence

recognition rates is much larger and this is largely due to the recovery of some function words such as 'the' and 'are' which had been previously deleted.

The following two tables shows the increased computation and storage that is required by the context sensitive algorithm using MSubs relative to that of the standard context free DP algorithm. The first table shows the results when phoneme level backtracking was used. Phoneme level backtracking is used to display the sequence of substitutions (including MSubs) along the optimal alignments. It was shown in chapter 7 that when the actual sequence of phoneme substitutions is required all of the nodes will have be transferred from the branch in the P-MSub tree into the DP matrix. However, template level backtracking can be used if we are only interested in the sequence of templates which make up the optimal alignment and in this case it not necessary to transfer the intermediate MSub nodes into the DP matrix.

Phoneme Level Backtracking	Rel. Computation	Rel. Storage
No MSubs	100%	100%
Word Internal MSub	237%	115%
Word Internal and Cross Word MSubs	274%	119%
All MSubs with Branch Score Caching	159%	119%

Figure 8–1: Relative Computation and Storage for Phoneme Level Backtracking

Template Level Backtracking	Rel. Computation	Rel. Storage
No MSubs	100%	100%
Word Internal MSubs	234%	106%
Word Internal and Cross Word MSubs	267%	110%
All MSubs with Branch Score Caching	152%	110%

Figure 8–2: Relative Computation and Storage for Template Level Backtracking

A comparison of the two tables shows that the requirement for phoneme level backtracking results in a very small amount of additional computation due to the

copying of paths into the DP matrix. A much larger increase in computation time is caused by the introduction of MSubs which span word boundaries. These are significantly more complicated to implement than word internal MSubs since these require the construction of pre-wb trees to hold each of the potential template hypotheses (see Ch 7.2), this complexity is reflected in the increased computation.

The final entry in each table shows that significant savings in computation could be achieved by caching the branch scores (Ch. 7.4.2). Branch score caching would reduce the amount of computation by calculating the DP alignment score of each branch just once after the pre-match pruning rather than recalculating the same scores every time the MSub is successfully matched against a template's reference frames. We have not had time to implement branch caching so the percentages which are shown are in fact estimates. The values were arrived at by bypassing the code which calculated the scores.

The following two examples are representative of the kind of effects which were obtained when applying positively correlated MSubs. In the first example the effect is positive, resulting in a correct alignment of the templates. However, in the second example the application of MSubs introduces a number of fortuitous matches along with a corresponding reduction in word recognition accuracy. In both of these examples the first alignment is that which was obtained by the usual DP algorithm using the standard context-free substitution penalties. The second alignment is the output of the DP algorithm using Msubs. The correlated sequence of phoneme substitutions (A.B) which make up the P-MSub are enclosed by square brackets. Each of the sequences [(A.B)...(Y.Z)] correspond to a single P-MSub.

8.4.1 A Good Example

In this example the application of Msubs has resulted in the uncovering of the correct transcription. In the first alignment the phonemes for the missing initial word template 'the' have been completely deleted due to the addition of the constant word offset which is added to the start of each new word hypothesis. Reducing

this penalty would allow the template 'the' to be recovered but at the expense of increasing the likelihood of additional fortuitous matches such as 'poor'.

```
((PATJLSB043 the preparation contains an undifferentiated carcinoma)
((   0  840  4517)      preparation      (-.dh)(-.@)(p.d)(r.r)(e.@)(p.d)(@.-)(r.r)(ei.ei)(sh.sh)
                                         (@.-)(n.n))
(( 840 1435  7470)         contains (#.-)(k.g)(@.@)(n.n)(t.t)(-.@)(ei.ei)(n.n)(z.z))
((1435 1510  8627)               an (#.-)(@.@)(n.n))
((1510 2970 18481) undifferentiated (#.-)(uh.uh)(n.n)(d.d)(i.i)(f.f)(@.-)(r.r)(e.e)(n.n)(sh.sh)
                                         (i.i)(-.y)(ei.i)(t.t)(-.y)(@.-)(d.d)(-.v)(-.o)(-.@)(-.z))
((2970 3375 21097)             poor (#.-)(p.d)(u@.uu)(-.ng)(-.v)(-.@)))

((PATJLSB043 the preparation contains an undifferentiated carcinoma)
((   0   85   928)              the       [(dh.dh)(@.@))
((  85  840  4321)      preparation (#.-)](p.d)(r.r)(e.@)(p.d)(@.-)(r.r)(ei.ei)(sh.sh)(@.-)(n.n))
(( 840 1435  6373)         contains [(#.-)(k.k)(@.@)(n.n)(t.t)](-.@)[(ei.ei)(n.n)(z.z))
((1435 1510  6830)               an (#.-)][(@.@)(n.n)])
((1510 2580 13845) undifferentiated (#.-)(uh.uh)(n.n)(d.d)(i.i)(f.f)(@.-)(r.r)(e.e)(n.n)(sh.sh)
                                         (i.i)(-.y)(ei.i)(t.t)(-.y)(@.-)[(d.d))
((2580 3375 18423)        carcinoma (#.-)](k.k)(-.o)(-.@)(aa.-)[(s.s)(i.-)](n.n)(ou.ou)(m.m)(-.v)
                                          (@.@)))
```

The word initial 'the' has been recovered in the second alignment because the correction factor of the MSub [(dh.dh)(@.@)(#.-)] outweighs the additional word offset that occurs as a result of introducing an additional word into the transcription.

The MSub [(s.s)(i.-)] could correspond to either human or machine phonology since /k aa s n ou m @/ is a plausible pronunciation of the word 'carcinoma'. The MSub [(d.d)(#.-)] is probably artifact of the limited training sentence data and states that a (d.d) substitution is particularly common before a word boundary (or equivalently, a word boundary deletion (#.-) typically occurs after a (d.d) substitution). Applying this MSub implements a kind of weak stochastic grammar which in this case results in the correct segmentation of the template 'undifferentiated' which had previously gone onto delete four additional phonemes that should have been substituted with the template 'carcinoma'

There are also a number of redundant MSubs such as [(#.-)(k.k)(@.@)(n.n)(t.t)] and [(ei.ei)(n.n) (z.z)(#.-)] which, in this case at least, serve no useful purpose in

terms of the final alignment and sequence of templates. For example, the sequence of phoneme substitutions in both instances of the template 'contains' is identical. The application of the MSub [(ei.ei)(n.n)(z.z)(#.-)] did not change the templates in the final transcription nor did it improve the quality of the alignment of 'contains'. The only result of applying this MSub has been to reduce the alignment score of 'contains' from 7470 to 6373. However, as we are usually only interested in the final sequence of templates, rather than the actual score of the alignment this is not particularly useful. The computational and storage burden of MSubs could be reduced if globally redundant Msubs such as these were pruned the trees.

8.4.2 A Bad Example

In the alignment shown below the introduction of MSubs results in a degraded word recognition score. In this example the phrase initial template for the word 'the' has also been completely deleted for the same reasons as in the first example.

```
((PATJLSB044 the preparation is overgrown)
((   0  820  5788)     preparation      (-.dh)(-.@)(p.d)(r.r)(e.o)(p.d)(@.-)(r.r)(-.@)(ei.ii)
                                        (sh.sh)(@.-)(n.n))
(( 820  960  6591)              is (#.-)(i.i)(z.z))
(( 960 1725 11417)       overgrown (#.-)(ou.ou)(v.v)(@.@)(g.ng)(-.@)(r.r)(ou.uu)(n.n)(-.ng)))

((PATJLSB044 the preparation is overgrown)
((   0  110   941)             the        [(dh.dh)(@.@))
(( 110  385  3337)            poor ( #.-)](p.d)(-.@)(u@.uu)(-.dh))
(( 385  615  4313)             red ( #.-)[(r.r)(e.e)](d.i))
(( 615  820  5251)            show [( #.-)(sh.sh)](ou.n))
(( 820  960  6054)              is ( #.-)(i.i)[(z.z))
(( 960 1725 10746)       overgrown ( #.-)](ou.ou)(v.v)(@.@)(g.ng)(-.@)(r.r)(ou.uu)(n.n)(-.ng)))
```

The application of P-MSubs once again recovers the phrase initial 'the' for the same reasons as discussed previously. However, the previously correct recognition of 'preparation' has now been mis-recognised as 'poor red show' due to fortuitous matches. The correction factors of the P-MSubs which were applied in the words 'red' and 'show' more than compensated for the word offset scores of the two additional words so the DP algorithm replaced 'preparation' with three shorter templates in the optimal (i.e. lowest scoring) alignment.

The cause of this disastrous mis-segmentation can be traced to the single MSub [(#.-)(sh.sh)] in 'show'. This MSub arose because most (sh.sh) substitutions were found to occur in word initial positions in the training sentences. In this case the correct position of the (sh.sh) substitution is obviously not word internal and the application of the MSub causes the wrong segmentation. If this MSub had not been applied the correct transcription would have been obtained. Although the application of this MSub produce a worse alignment in this exampler there could be other situations where its application could be beneficial. The MSub [(r.r)(e.e)] in 'red' does not contribute to this mis-segmentation since it is easy to see that this MSub would also have been present in the alignment of 'preparation', replacing the previous sequence (r.r)(e.o) of substitutions. This effect of this MSub has been to correct the output of the AFE which appears to have incorrectly ranked /o/ above /e/ in that input segment.

The additional fortuitous matches could be eliminated by further increasing the word offset score so that the score offsets which arise from the creation of the two additional words outweighs the score reduction due to the correction factor of the MSub. Unfortunately this will also lead to many more function words such as 'the' being deleted although in theory it should still be possible to recover these function words by increasing the scores of MSubs such as (dh.dh)(@.@)(#.-). It would also be relatively easy to create a more restrictive grammar which for example forces sentences to begin with 'the' whilst disallowing phrases such as 'poor red show' which do not occur in any of the sentences in our cytology domain. However, as we increase the restrictiveness of the grammar the perplexity decreases and recognition task becomes easier since there are less template candidates to choose amongst. In this situation there is less potential for MSubs to have any effect at all, either good or bad.

8.4.3 An Optimal P-MSub Tree

The most successful way of eliminating fortuitous matches, both in terms of increasing recognition accuracy and decreasing computation, is to simply prune the

offending MSubs from the tree. Many of the MSubs in the trees which are currently generated appear to serve no useful purpose at all. These MSubs are applied in many situations where they should not have been applied, producing incorrect alignments, and in the situations where they really should have been applied the correct alignment would have been obtained whether or not the MSub was applied. Identical or better results would be obtained if these MSubs were simply pruned from the trees. The computational overheads would also be reduced since the pruned trees will be smaller and can therefore be matched quicker with the lattice segments and template reference frames.

An optimal P-MSub tree was constructed by hand in order to determine the best recognition performance that could be reasonably expected. This tree was initially constructed by pruning all nodes from the original tree which represented MSubs that served no useful purpose or represented MSubs whose negative effects outweighed any positive effects. The pruned tree was then given to the context sensitive DP algorithm and used to align the set of 60 phoneme lattices which had been previously used for training. These alignments were then compared with the original correct alignments that had been obtained during training.

The outcome of this comparison suggested a number of improvements. New MSubs were added whose application would hopefully uncover additional correct alignments. The correction factors of existing MSubs were also modified to either increase the number of missing correct applications or decrease the number of fortuitous incorrect application. Any MSubs which had become redundant due to previous modifications were also pruned from the tree. The alignments were then regenerated using the new tree and further modifications were made as deemed necessary. This process was repeated until no obvious improvements could be found. The optimal MSub tree which was produced by this process is shown below.

The optimal P-MSub tree (fig. 8–3 contains 22 nodes compared with an average of 211 nodes in the original P-MSub trees and contains just one word boundary spanning MSub. Many of correction factors stored on the nodes of this tree are zero. The MSubs which correspond to these nodes are ignored by the P-MSub

application algorithms since they would not change the original alignment scores. The optimal MSub tree therefore contains just 9 applicable MSubs. The word and sentence recognition rates obtained using the P-MSub and pre-wb trees which were derived from this tree were 72% and 25% respectively with no grammar. These comfortably exceed the best recognition rates that were obtained using the original MSub trees. These results show that there is considerable potential for refining the current tree decomposition algorithm. In the section of this chapter outlining possible directions for future work the use of corrective training techniques are suggested which might automate this process.

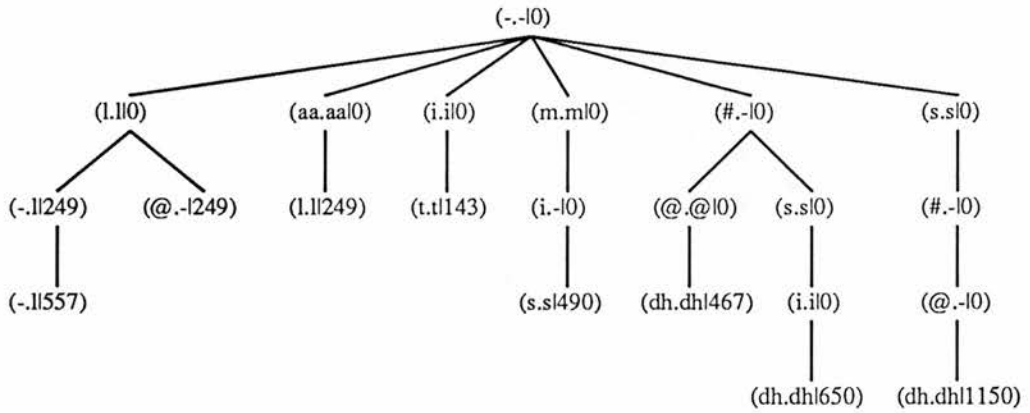


Figure 8–3: A hand crafted optimal P-MSub Tree

The three MSubs [(dh.dh)(@.@)(#.-)], [(dh.dh)(i.i)(s.s)(#.-)], and [(dh.dh)(@.-)(#.-)(s.s)] are used to ensure that the phrase initial alignments of the templates 'the' and 'this' are obtained. The last of these MSubs was obtained by appending an (s.s) substitution to the MSub [(dh.dh)(@.-)(#.-)] which was present in the initial version of the the optimal tree. Although this MSub helps by introducing weak syntactic constraints the principle reason for the addition of the (s.s) substitution was to reduce the over application of the original MSub. The original MSub was applied whenever the phoneme /dh/ occurred in the lattice and this led to the presence of a number of fortuitous matches of 'the'. The fortuitous matches were eliminated by appending the (s.s) substitution. The lattice is now required to contain the phoneme sequence /dh s/ and there are also constraints

on those templates that can follow any (fortuitous) alignments since these must now contain a word initial /s/ phoneme.

The MSub [(-.l)(-.l)(l.l)] helps to ameliorate the tendency of the AFE to mis-segment /l/ phonemes. The MSub [(i@.i)(-.@)] was not included since this mis-segmentation did not cause any problems for the lexical access algorithm. The remaining MSubs were chosen by comparing the training and evaluation alignments of the same lattices. MSubs were chosen which would produce scores for the correct alignments generated during training below those of the incorrect alignments being generated during evaluation.

8.4.4 A Comparison With Other Approaches

In chapter 2 we described four alternative approaches of incorporating phonological knowledge which have been used with varying degrees of success. Of these four approaches phonological parsing (Ch. 2.3) requires recognition performance which is beyond current acoustic front ends and lattice expansion (Ch. 2.1) encountered serious problems which led to its abandonment. Implementations of the two remaining approaches have been more successful. Pre-compilation (Ch. 2.2) applies the phonological rules off-line in their usual generative direction to base-form entries in the lexicon to create a complex network of possible pronunciations. In contrast speech recognition systems which use finite state machines (FSMs) encode the phonological knowledge in a separate structure which simplifies the lexicon at the expense of increased run-time computation. These last two approaches are compared with one another along with our approach which involves using trees of correlated macro substitutions to apply phonological knowledge at run-time.

Each of the alternative paths in a pre-compiled network represent possible variations in the pronunciations. These paths correspond in many ways to the paths through a FSM or branches in our MSub trees which also describe variations in pronunciations. In the example shown in fig. 2–2 in chapter 2 the phonological rules /d # y/ $\mapsto$ /j/ and /n d # l/ $\mapsto$ /n # l/ were used to compile a lexi-

con containing a small number of word templates. The application of these rules created additional paths which represented the possible variations in pronunciation. These rules can also be rewritten in the form of MSubs as [(d.j)(#.-)(y.y)] and [(n.n)(d.-)(#.-)(l.l)] or, by using different terminology, in the form of a path through a FSM as d:j,#:-,y:y and n:n,d:-,#:-,l:l. The first component X of each substitution (X.Y)/X:Y corresponds to phonemes along the path representing the original base form pronunciation whilst the second component Y corresponds to phonemes along the path representing the alternative pronunciation.

During precompilation the application of the rule /d # y/ $\mapsto$ /j/ inserts an additional path into the network wherever the sequence of phonemes /d # y/ is found to occur. The size and complexity of the network is therefore proportional to the product of the number of base form pronunciations and the number of phonological rules which can rapidly become impracticable. In the case of FSMs and MSubs each phonological rule corresponds to a single path or branch and there is no duplication of paths or branches for each base form pronunciation. By using these approaches the lexicon can contain just the base form pronunciations and has a size and complexity which is independent of the number of phonological rules. Unfortunately it is now necessary to identify the locations at which the alternative paths are applicable since this information is no longer present in the lexicon. So although FSMs and MSubs enable the phonological rule base to be decoupled from the lexicon a price has to be paid in the form of additional run time computation.

The paths corresponding to alternative pronunciations are propagated by a DP algorithm through the lexical network or the FSM in parallel with paths corresponding to the original base form pronunciations and in an identical manner (i.e. by forward scanning described in Chapter 7). Information such as accumulated distance scores and backtracking pointers must be stored on intermediate points along the paths. This information is required by the DP algorithm in order to propagate partial alignments to the next point along each path. The run-time storage which is required to preserve this information is in both cases proportional to the number of alternative paths which are being propagated by the DP algo-

rithm. As before this is proportional to the product of the number of templates and the number of phonological rules. So, although the use of FSMs reduces the initial (i.e. off-line) size and complexity of the lexicon a similar amount of run-time storage will be required when using either pre-compilation techniques or FSMs.

Rather than propagating alternative paths in parallel with the base form paths we first identify alternative paths which could be used to generate alternative pronunciations and then use the DP algorithm to calculate the alignment scores of these paths. The alignment scores of identified paths (i.e. branches in an MSub tree) are then calculated in a single step instead of being calculated step by step, time synchronously, from left to right as is the case with the pre-compiled networks and FSMs.

Applicable paths are identified by comparing the template and input components of each substitution in an MSub in reverse with successively earlier lattice segments and template reference frames. As all of the past reference frames and lattice segments are known it is always possible to identify, given the current contents of the lattice and template reference frames, those MSubs whose path scores should have been calculated. There is therefore no need to preserve accumulated distance scores and backtracking pointers which were previously needed to be propagate paths in the following stages of the DP algorithm. The additional run-time storage which is required for this approach is very small and is also independent of the size of the lexicon.

The following table summarises the differences in the computational and storage requirements of these approaches.

	Pre-Compilation	FSMs	MSubs
Computation	$\propto K * R * J$	$\propto K * R * J$	$\propto K * R * J$
Off-line Storage	$\propto K * R * J$	$\propto K + R * J$	$\propto K + R * J$
On-line Storage	$\propto L * R * J$	$\propto K * R * J$	$\propto K + R * J$

where K = number of templates

R = number of rules

J = average rule length

In the case of FSMs or MSubs the term $R * J$ represents the size of the FSM or MSub tree respectively. However these terms are not strictly comparable since the FSM network would be typically smaller than an equivalent MSub tree containing the same set of phonological rules. The FSM networks can contain PROLOG type variables which can be matched with many different phonemes and this allows certain types of rules such as word boundary assimilations to be expressed by a single path through the network. In contrast MSub trees do not contain variables and therefore each possible instantiation of the variable would have to be represented by a separate branch in the tree. However the use of fully specified phonemes does allow many of these branches to be pruned from the tree. Therefore the fact that the MSub trees are larger than their equivalent FSM networks is unlikely to be reflected in increased computation times.

An additional and important advantage of our technique is the ability to automatically extract MSubs by statistically analysing previous alignments. This eliminates the need for human intervention in formulating phonological rules, translating these rules into formats usable by the algorithms, and determining which rules should be used and which can be omitted. The rules which are extracted arise from a number of sources which include human phonology and machine phonology as well as artifacts of the language (syllable structure) and application domain (common words/phrases). It would be an extremely time consuming and tedious operation to extract equivalent rules by hand since many of these rules are dependent upon the application domain and the performance characteristics of the acoustic front end.

8.5 Future Work

8.5.1 Removing Redundant MSubs

As we have seen the trees contain many macro-substitutions which, although valid, are in some sense redundant or undesirable. Many of the MSubs appear to serve no useful purpose (assuming that we are more interested in the sequence of templates

than the details of the alignment) whilst others actually produce worse alignments. If the application of an MSub never causes a positive change in the final alignment then the macro-substitution is effectively redundant. Identical or better alignments would still be obtained if these MSubs were pruned from the tree and the context-sensitive DP alignment process would also be quicker. It should be possible to automatically detect redundant MSubs by comparing the correlation trees that would by generated by Treelab from the alignments obtained during training and evaluation.

We envisage applying Treelab to the alignments that would be obtained from the training and evaluation processes using the same sets of phoneme lattices. Both sets of alignments should be calculated by the context-free DP algorithm which does not use MSubs. Ideally the alignments and templates sequences that would be produced by the training and evaluation processes would be identical since the phoneme lattices are the same in each case. Inevitably the recognition rates will be less than perfect and there will be differences between the two sets of alignments and the trees that are generated from them. If we recall that the effect of MSubs is to modify the probabilities of the sequences that they represent then we can identify the redundant MSubs as being those which have similar probabilities and correction factors in both trees. These MSubs can be considered redundant since the sequences which they represent are being observed with the correct probability in the evaluation data, there is therefore no need for the MSubs which modify the probability of these sequences.

8.5.2 Corrective Training

As we have seen in previous chapters there are problems with interpreting the substitution probabilities obtained during training as probabilities when used in evaluation. This led us to propose a form of corrective training which was applied to the substitution penalty matrices which adjusted the penalties in an attempt to equalise the observation frequencies of the phoneme substitutions in the training and evaluation alignments. It should be possible to extend this technique to

the MSub trees and implement a higher order corrective training algorithm which attempts to equalise the observation frequencies of substitution sequences (i.e. MSubs) in addition to individual substitutions. The technique of corrective training was based upon the premise that if the substitution frequencies are similar then the alignments are also more likely to be similar thus giving higher word and sentence recognition scores. It will be even more likely that the alignments will be similar if the probabilities of longer sequences are also similar so we would expect the corrective training of MSubs to have an even greater impact on the recognition scores.

The basic techniques could be virtually identical to that which have been previously used (Ch. 4.1) although we would now be comparing trees of substitution sequences rather than confusion matrices. Treelab would be used to construct the trees from the alignments produced by the context-free DP alignment process during training and evaluation. Each of the sequences in these two trees would then have to be compared in order to determine those sequences of phoneme substitutions which occurred more often during training than during evaluation and vice-versa. Correction factors would then be calculated based upon these relative frequencies which would decrease the DP penalty of those sequences which did not occur often enough and increase the penalty of those sequences which occurred too often. As before it is likely that a number of passes would be required before an optimum set of correction factors was obtained.

8.5.3 Automatic Speaker Adaptation

One of the advantages of this approach is the ease with which multiple trees can be handled. It is possible in principle to have a number of trees active at the same time and delay the choice of the active trees or change the choice at run-time. This would allow a hierarchy of trees to be used whose components are dynamically chosen at run-time. The top-level tree would contain the most general MSubs which are generally applicable, this would include sequences that account for the machine phonology of the AFE and speaker independent human

phonology. Lower level trees would contain MSubs that accounted for phonological phenomena that were specific to progressively smaller classes of speakers, possibly down to individual speakers.

Given sufficient training data it should be possible to automatically construct each of the trees containing speaker independent and progressively speaker dependent MSubs. In order to do this we would need to build a number of trees from the alignments between the training sentences and the lattices produced by the AFE when processing the speech of each speaker. Each of these trees would be constructed in an identical manner to those which we have been currently using and as we have seen they we will contain a mixture of machine phonology and human phonology (both speaker independent and dependent) as well as various syntactic artifacts. It should also be possible to automatically determine the appropriate trees by either choosing the combination of trees which give the best performance on a set of known sentences, or by using Treelab to analyse the alignments at run-time and choosing the tree which best explain the sequences in these alignments.

We could create the trees containing general MSubs that are applicable to each speaker as well as those which are specific to certain speaker(s) by comparing branches in the tree to find sequences which have similar or significantly different probabilities. If all of the speakers spoke the same set of training sentences, or each set of training sentences was sufficiently representative of the domain, we could expect that the branches corresponding to machine phonology, speaker independent human phonology and syntactic artifacts would have similar scores in each of the trees. Therefore, a tree containing these general speaker independent MSubs could be generated from those MSubs which were common to each of the trees and had similar scores. Once a tree of speaker independent MSubs has been created the trees of speaker dependent MSubs can be created by pruning the branches in the original trees which are also present in the speaker independent tree. This should remove all branches corresponding to machine and speaker independent phonology as well as syntactic artifacts leaving those MSubs which represent phonological phenomena that are particular to each of the speakers.

8.5.4 Broad-class Phonemes

It might be desirable to generalise the phonemes used in the MSubs by introducing broad class phonemes such as nasals, front vowels, plosives etc. It is well known that many phonological rules are conditioned by various phonetic features, like place of articulation, which are common to a number of phonemes. At present each of the corresponding MSubs would have to be expressed individually which results in trees which are larger than necessary and require more training data in order to reliably estimate the correction factors. Although the routines which we have used to identify and apply MSubs can handle underspecified phonemes we currently have no means of automatically generating such MSubs.

However, even if we were able to generate trees of MSubs containing broad class phonemes it seems unlikely that they would have any beneficial consequences for the implementation procedures. Although the trees would be smaller there will be less scope for pre-match pruning (Ch. 7.3) since the likelihood of a broad class specifier matching the phonemes in an input segment is much higher. Also, presence of generalised phonemes on the nodes will prevent the caching of the alignment scores of the branches (Ch. 7.4.2) since the alignment along each branch will not be fully specified until the branch has been successful matched against a template. The presence of broad class phonemes would increase the complexity of the matching procedures and waste much time by recalculating the alignment scores of each branch. Therefore, even if the trees did contain broad class phonemes it would probably be advantageous to expand the phonemes and generate separate branches for each possible phoneme instance before the trees were used by the DP algorithm.

8.6 Conclusion

In this thesis we have described a novel technique for efficiently incorporating phonological knowledge into the lexical access component of a speech recognition system. In contrast to alternative techniques which propagate paths corresponding to various pronunciations in the forward direction, time synchronously and in parallel with the paths corresponding to the base form pronunciations we identify potential paths in reverse and then calculate scores for applicable paths in a single step. This approach eliminates the need to maintain large amounts of temporary storage. The amount of additional storage which is required to implement the MSubs is constant (when template level backtracking is required) and is independent of the size of the lexicon. The increase in computation is also relatively modest requiring approximately 152% more when using an MSub tree containing an average of 211 nodes virtually all of which correspond to a potential phonological rule. In comparison with existing approaches this method seems most likely to be able to scale for use in the lexical access component of large vocabulary speaker independent recognition systems.

Macro-substitutions are automatically extracted from previous alignments with no manual intervention. The rules are extracted by algorithms which statistically analyse previous recognitions and look for correlated sequences of substitutions which will arise from the application of phonological rules. The MSubs which are extracted are necessarily compatible with the inventory and discriminatory capabilities of the AFE so no tedious postprocessing of the rules is required. Although more training data would allow more accurate and subtler phonological phenomena to be detected the amount of training data does not seem to be the limiting factor in the recognition performance. We have shown by hand constructing an optimal MSub tree that significant increases in recognition accuracy can be achieved by pruning most of the branches from the trees. The problem is determining which MSubs can be discarded rather than extracting more MSubs which represent subtler phonological phenomena.

Appendix A

A.1 The Machine Readable Phonemic Alphabet (MRPAbet)

Vowels		Diphthongs		Consonants	
/i/	bid	/ei/	day	/p/	pea
/ii/	bead	/ou/	go	/b/	bea
/e/	bed	/au/	cow	/t/	chew
/a/	bad	/ai/	eye	/d/	dye
/aa/	bard	/oi/	boy	/k/	key
/uh/	bud	/i@/	beer	/g/	guy
/@@/	bird	/e@/	bare	/m/	me
/@/	the	/u@/	tour	/n/	name
/o/	pot			/ng/	sing
/oo/	port			/f/	fan
/u/	put			/v/	van
/uu/	boot			/th/	thin
				/dh/	then
				/s/	sea
				/z/	zoo
				/sh/	she
				/ch/	chew
				/jh/	judge
				/h/	hat
				/w/	way
				/y/	yes
				/l/	lay
				/r/	ray

A.2 The Chart Parser Grammar

The Chart Parser Grammer used for the 80 Cytology Utterances

top → s	Valid phrases are sentences or noun phrases
top → np	
s → np, vp	A sentence is a noun phrase followed by a verb phrase
vp → v, np	A verb phrase is a verb followed by a noun phrase
vp → v, pp	or prepositional phrase
vp → be, np	A 'be' verb (is, are etc.) can precede a noun phrase
vp → be, adj	or adjective
np → n2	A noun phrase is a compound noun which may be preceded
np → det, n2	by a determiner
pp → p, np	A prepositional phrase is a preposition followed by a noun phrase
det → art	A determiner is an article or quantifier
det → quan	
n2 → n	A compound noun is a noun optionally preceded by one or
n2 → adj, n	two adjectives
n2 → adj, adj, n	

A.3 Typical Confusion Matrices

These confusion matrices were generated from one partition of the 80 cytology sentences during maximum likelihood training. Vowel phonemes in the lexicon contain stress markers which label reduced, normal and stressed phonemes. However the AFE does not discriminate between reduced, unstressed, and stressed vowels. Only the matrices labelled 'No Stress' were used to calculate the substitution penalties. The values in the 'No Stress' matrices are equal to the sum of the values in the 'Reduced', 'Normal', and 'Stressed' matrices.

Conosonat - Vowel Substitution Counts : No Stress
X axis = Template phonemes, Y axis = Input phonemes.

	-	i	ii	e	a	aa	uh	@@	@	o	oo	u	uu	ei	ou	au	ai	oi	i@	e@	u@	##
-	0	19	0	0	17	0	5	9	1	0	5	0	2	0	0	1	0	0	0	0	0	0
t	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	3	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
ng	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
th	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dh	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
z	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
sh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ch	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
jh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	14	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
r	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0

Consonant - Consonant Substitution Counts : No Stress
X axis = Template phonemes, Y axis = Input phonemes.

	-	t	p	k	b	d	g	m	n	ng	f	v	th	dh	s	z	sh	zh	ch	jh	h	w	y	l	r
-	0	0	0	0	0	0	0	0	0	32	1	36	4	56	0	18	0	0	0	4	9	14	37	38	5
t	8	66	0	2	0	4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
p	0	0	27	3	13	7	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
k	0	0	0	33	0	3	24	0	0	3	0	0	1	0	2	0	0	0	0	0	6	0	0	0	0
b	0	0	0	0	14	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
d	0	0	0	0	0	26	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
g	0	0	0	0	0	0	3	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	72	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
n	3	0	0	0	0	7	0	0	119	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ng	1	0	0	0	0	0	0	1	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0
v	10	0	0	0	0	0	0	0	0	0	0	15	0	2	1	0	0	0	0	0	0	0	0	0	1
th	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
dh	3	0	0	0	0	0	0	0	6	0	0	2	0	34	0	0	0	0	0	0	0	0	0	0	0
s	1	0	0	0	0	0	0	0	0	0	0	0	1	2	122	0	0	0	0	0	0	0	0	0	0
z	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89	0	0	0	0	0	0	0	0	0
sh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0
zh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
ch	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
jh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0
h	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	1	0
y	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	22	0	0
l	14	0	0	0	1	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	6	7	69	0
r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44

Vowel - Vowel Substitution Counts : No Stress
X axis = Template phonemes, Y axis = Input phonemes.

	-	i	ii	e	a	aa	uh	@@	@	o	oo	u	uu	ei	ou	au	ai	oi	i@	e@	u@	##
-	0	19	0	0	17	0	5	9	1	0	5	0	2	0	0	1	0	0	0	0	0	0
i	20	60	29	0	0	0	0	0	6	0	0	2	5	0	10	0	0	0	4	0	0	0
ii	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	43	0	0	11	0	1	0	0	3	1	1	4	0	0	0	0	0	0	0
a	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aa	0	0	0	0	0	6	0	0	1	9	0	1	1	0	0	0	0	0	0	0	0	0
uh	0	0	0	0	0	4	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@@	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
@	37	35	0	2	0	0	0	0	112	4	10	0	1	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	2	0	0	2	18	1	1	0	0	0	0	0	0	0	0	0	0
oo	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0
u	1	0	3	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
uu	0	0	0	0	0	0	0	0	3	0	0	0	17	0	0	0	0	0	0	0	0	0
ei	1	1	0	0	0	0	0	0	0	0	0	0	0	39	0	0	0	1	0	0	0	0
ou	0	0	0	0	0	0	0	0	0	0	0	5	5	0	21	0	0	0	0	0	0	0
au	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ai	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0
oi	0	0	0	0	0	0	0	0	0	0	1	0	5	0	0	0	0	0	0	0	0	0
i@	0	1	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0
e@	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u@	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Vowel - Vowel Substitution Counts : Reduced
X axis = Template phonemes, Y axis = Input phonemes.

	-	i	ii	e	a	aa	uh	@@	@	o	oo	u	uu	ei	ou	au	ai	oi	i@	e@	u@	##
-	0	19	0	0	17	0	5	9	1	0	5	0	2	0	0	1	0	0	0	0	0	0
i	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
ii	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oo	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uu	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ei	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0
ou	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
au	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ai	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0
oi	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
i@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Vowel - Vowel Substitution Counts : Normal
X axis = Template phonemes, Y axis = Input phonemes.

	-	i	ii	e	a	aa	uh	@@	@	o	oo	u	uu	ei	ou	au	ai	oi	i@	e@	u@	##
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	18	44	27	0	0	0	0	0	4	0	0	2	4	0	5	0	0	0	2	0	0	0
ii	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	4	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aa	0	0	0	0	0	0	0	0	1	8	0	0	1	0	0	0	0	0	0	0	0	0
uh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@	37	35	0	2	0	0	0	0	112	4	10	0	1	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	1	0	3	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
uu	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
ei	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ou	0	0	0	0	0	0	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	0
au	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ai	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
oi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Vowel - Vowel Substitution Counts : Stressed
X axis = Template phonemes, Y axis = Input phonemes.

	-	i	ii	e	a	aa	uh	@@	@	o	oo	u	uu	ei	ou	au	ai	oi	i@	e@	u@	##
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	2	15	2	0	0	0	0	0	1	0	0	0	1	0	4	0	0	0	2	0	0	0
ii	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	39	0	0	11	0	0	0	0	3	1	0	4	0	0	0	0	0	0	0
a	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aa	0	0	0	0	0	6	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
uh	0	0	0	0	0	4	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@@	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	2	0	0	2	18	1	1	0	0	0	0	0	0	0	0	0	0
oo	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uu	0	0	0	0	0	0	0	0	3	0	0	0	15	0	0	0	0	0	0	0	0	0
ei	1	1	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	1	0	0	0	0
ou	0	0	0	0	0	0	0	0	0	0	0	4	5	0	15	0	0	0	0	0	0	0
au	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ai	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
oi	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0
i@	0	1	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0
e@	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u@	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Vowel - Consonant Substitution Counts : No Stress
X axis = Template phonemes, Y axis = Input phonemes.

	-	t	p	k	b	d	g	m	n	ng	f	v	th	dh	s	z	sh	zh	ch	jh	h	w	y	l	r
-	0	0	0	0	0	0	0	0	0	32	1	36	4	56	0	18	0	0	0	4	9	14	37	38	5
i	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ii	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@	37	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	7	6
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uu	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ei	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ou	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
au	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ai	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Vowel - Consonant Substitution Counts : Reduced
X axis = Template phonemes, Y axis = Input phonemes.

	-	t	p	k	b	d	g	m	n	ng	f	v	th	dh	s	z	sh	zh	ch	jh	h	w	y	l	r
-	0	0	0	0	0	0	0	0	0	32	1	36	4	56	0	18	0	0	0	4	9	14	37	38	5
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ii	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uu	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ei	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ou	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
au	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ai	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Vowel - Consonant Substitution Counts : Normal
X axis = Template phonemes, Y axis = Input phonemes.

	-	t	p	k	b	d	g	m	n	ng	f	v	th	dh	s	z	sh	zh	ch	jh	h	w	y	l	r
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ii	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@	37	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	7	6
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uu	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ei	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ou	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
au	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ai	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

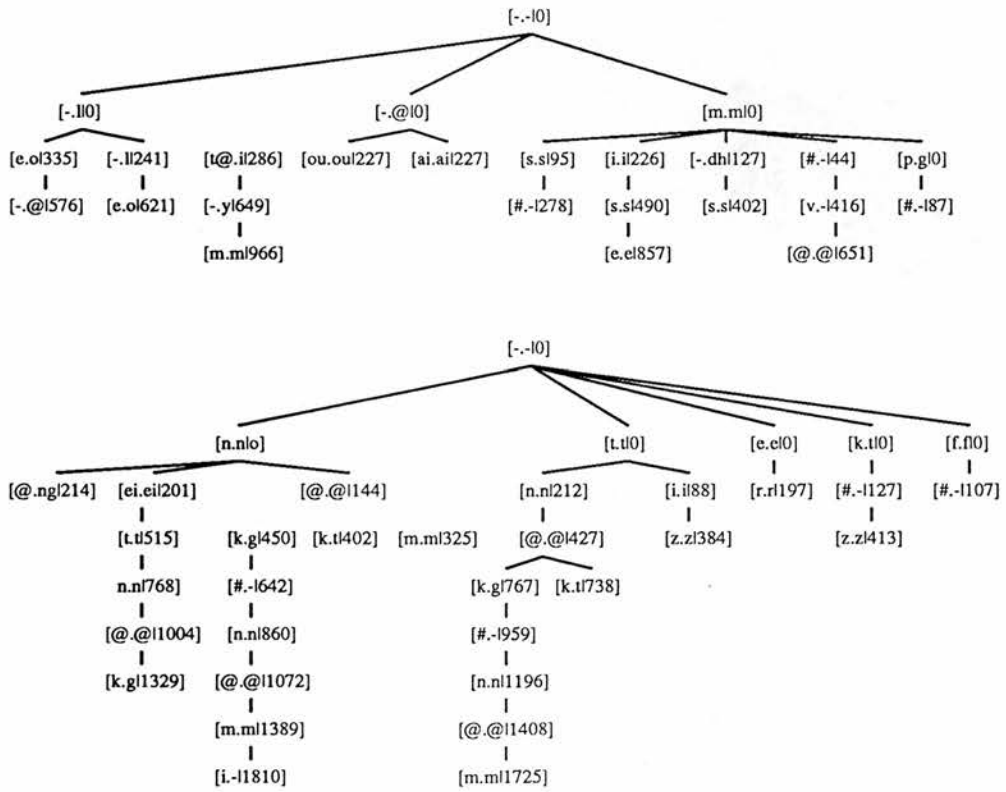
Vowel - Conosonant Substitution Counts : Stressed
X axis = Template phonemes, Y axis = Input phonemes.

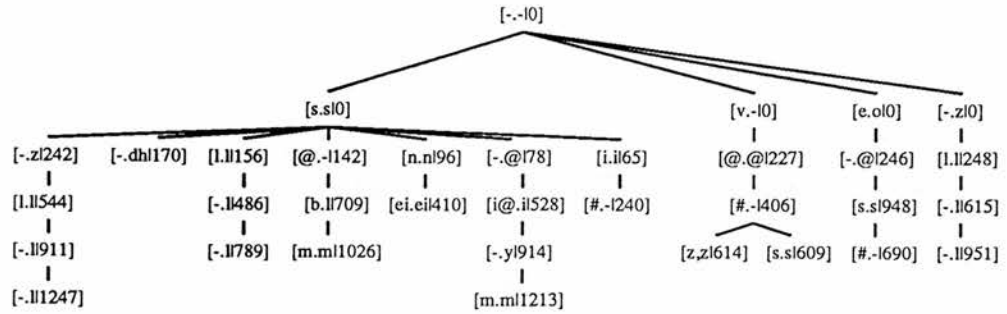
	-	t	p	k	b	d	g	m	n	ng	f	v	th	dh	s	z	sh	zh	ch	jh	h	w	y	l	r
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ii	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
aa	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uh	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
uu	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ei	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ou	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
au	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ai	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
oi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
u@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

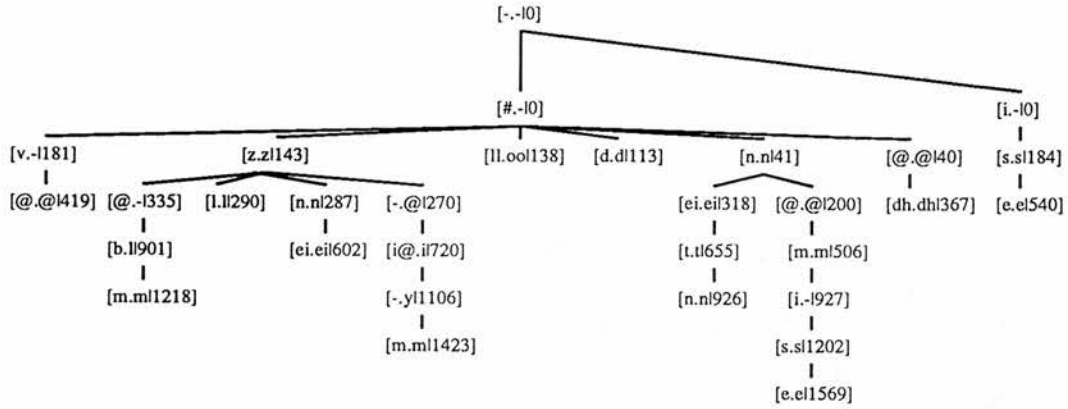
A.4 Sample Macro-Substitution Trees

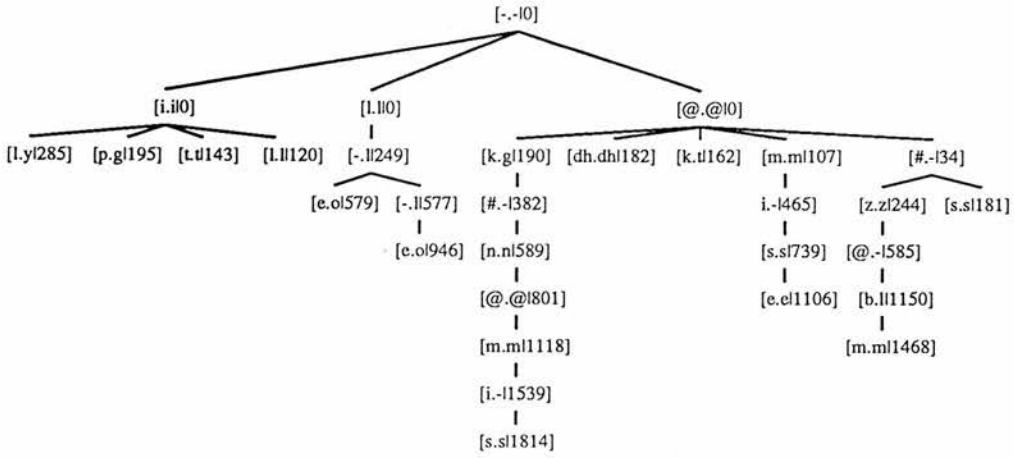
These macro-substitution trees were automatically extracted from phoneme substitution sequences generated during maximum likelihood training on one partition of the 80 cytology sentences. The P-MSub tree has been broken up into a number of smaller trees which can fit on the page. The actual P-MSub tree would be recreated by merging the root nodes of each of the P-MSub sub-trees.

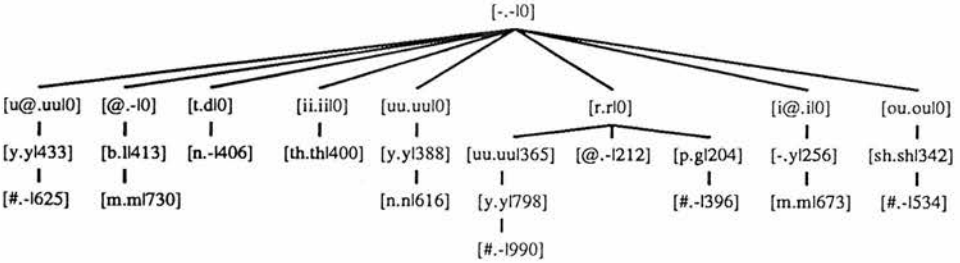
A.4.1 P-MSub Tree



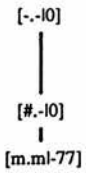

[-.-|0]
[s.s|0]
[v.-|0]
[e.o|0]
[-.z|0]
[-.z|242]
[-.dh|170]
[l.l|156]
[@.-|142]
[n.n|96]
[-.@|78]
[i.i|65]
[@.@|227]
[-.@|246]
[l.l|248]
[l.l|544]
[-.l|486]
[b.l|709]
[ei.ei|410]
[i@.i|528]
[#.-|240]
[#.-|406]
[s.s|948]
[-.l|615]
[-.l|911]
[-.l|789]
[m.m|1026]
[-.y|914]
[z,z|614]
[s.s|609]
[#.-|690]
[-.l|951]
[-.l|1247]
[m.m|1213]


[-.-|0]
[#.-|0]
[i.-|0]
[v.-|181]
[z.z|143]
[l.l|oo|138]
[d.d|113]
[n.n|41]
[@.@|40]
[s.s|184]
[@.@|419]
[@.-|335]
[l.l|290]
[n.n|287]
[-.@|270]
[ei.ei|318]
[@.@|200]
[dh.dh|367]
[e.e|540]
[b.l|901]
[ei.ei|602]
[i@.i|720]
[t.t|655]
[m.m|506]
[m.m|1218]
[-.y|1106]
[n.n|926]
[i.-|927]
[m.m|1423]
[s.s|1202]
[e.e|1569]

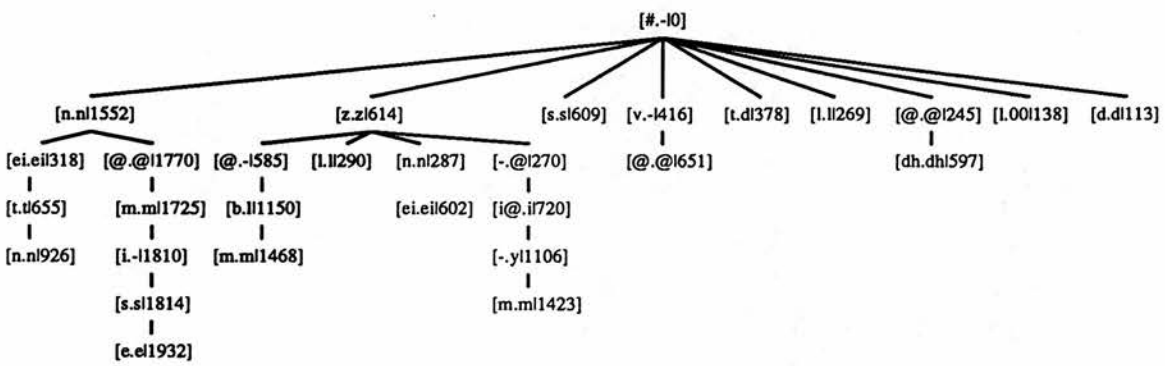

[-.-|0]
[i.i|0]
[l.l|0]
[@.@|0]
[l.y|285]
[p.g|195]
[t.t|143]
[l.l|120]
[-.l|249]
[k.g|190]
[dh.dh|182]
[k.t|162]
[m.m|107]
[#.-|34]
[e.o|579]
[-.l|577]
[#.-|382]
i.-|465]
[z.z|244]
[s.s|181]
[e.o|946]
[n.n|589]
[s.s|739]
[@.-|585]
[@.@|801]
[e.e|1106]
[b.l|1150]
[m.m|1118]
[m.m|1468]
[i.-|1539]
[s.s|1814]


[-.-|0]
[u@.uu|0]
[@.-|0]
[t.d|0]
[ii.ii|0]
[uu.uu|0]
[r.r|0]
[i@.i|0]
[ou.ou|0]
[y.y|433]
[b.l|413]
[n.-|406]
[th.th|400]
[y.y|388]
[uu.uu|365]
[@.-|212]
[p.g|204]
[-.y|256]
[sh.sh|342]
[#.-|625]
[m.m|730]
[n.n|616]
[y.y|798]
[#.-|396]
[m.m|673]
[#.-|534]
[#.-|990]

A.4.2 N-MSub Tree



A.4.3 Pre-wb Tree



A.5 Results

This section shows the phoneme level bactracking output of the context sensitive DP alignment algorithm. The DP algorithm was trained using 60 of the 80 cytology sentences and lattices and the alignments of the training sentences were used to construct the MSub trees. The remaining twenty lattices were then optimally aligned with the 206 templates in the lexicon. Grammatical constraints were provided by the chart parser using the grammar listed in app. A.2. The alignments which were generated by the lexical access algorithm are shown below. Each alignment shows the correct sentence and the sequence of words which were recognised. The first item in the sequences shows the starting time, finishing time, and accumulated distance score of the alignment through the template. Correlated sequences of phoneme substitutions (i.e. MSubs) are enclosed within square brackets, e.g. [(y.y)(uu.uu)].

```
((PATJLSB003 a review confirms the inflammatory nature)
((   0  530  3040)          review (-.w)(r.r)(i.i)(v.v)(-.y)(-.y)[(y.y)(uu.uu)])
(( 530 1200  8438)        confirms [(#.-)(k.k)(@.@)(n.n)](f.f)(@@.ou)(-.o)(m.m)(-.l)[(z.z))
((1200 1330  9864)           thick (#.-)](th.th)(i.i)(k.-))
((1330 2050 14602)    inflammatory (#.-)(i.i)(n.n)(f.f)(l.l)(a.a)(m.m)(@.-)(t.-)(r.@)(-.jh)(i.i))
((2050 2585 17723)          nature (#.-)(n.n)(-.@)(ei.ei)(ch.ch)(-.@)(@.@)))


((PATJLSB007 groups of malignant cells are present)
((   0  470  2007)          groups (g.g)(r.r)(uu.uu)(p.d)(s.s))
(( 470 1345  7384)        contains (#.-)](k.-)[(@.@)(n.n)](t.-)[(-.l)(-.l)][(eii.ei)(n.n)](-.@)
                                        (-.dh)(-.dh)[(z.z)(-.@))
((1345 2255 12255)         present (#.-)](-.o)(-.l)(-.z)(-.@)(p.d)(r.r)(e.e)(z.z)[(@.@)(n.n)
                                        (t.t)]))


((PATJLSB011 microscopy shows cellular smears)
((   0  830  3999)      microscopy (m.m)(ai.ai)(k.k)[(r.r)(o.o)](s.s)(k.k)[(@.@)(p.d)](-.y)(i.i))
(( 830 1130  5015)          showed [(\#.-)(sh.sh)(ou.ou)](d.uu))
((1130 1680  7131)        cellular (\#.-)(s.s)(e.e)(l.l)(y.y)(u.@)(l.l)(@.@))
((1680 2405  9574)          smears [(\#.-)(s.s)(-.dh)][(m.m)(-.y)(i@.i)(-.@)](-.@)[(-.dh)(z.z)]))
```

```
((PATJLSB015 numerous sheets of epithelial cells)
((   0  505  1880)        numerous [(n.n)(y.y)(uu.uu)](m.m)(@.-)(r.r)(@.i)[(s.s))
(( 505  905  3623)          sheets (\#.-)](sh.sh)(ii.ii)(t.g)[(s.s))
(( 905 1225  5659)          appear (\#.-)](-.v)[(@.@)(p.d)](i@.i))
((1225 1670  8334)            four [(\#.-)(f.f)](-.y)(-.y)(-.@)(oo.oo))
((1670 2375 11481)           cells [(\#.-)(s.s)](-.l)[(e.o)(-.l)][(-.l)(-.l)][(l.l)(-.z)(z.z)]))


((PATJLSB019 small sheets of malignant cells are present)
((   0  410  1856)           small (s.s)(m.m)(oo.oo)(l.w)(-.@))
(( 410 1275  6024)          sheets [(\#.-)(sh.sh)](ii.ii)(t.g)(s.s)(-.@)[(-.l)(-.l)](-.y)[(-.@))
((1275 1885 10886)             are (\#.-)](-.dh)(-.ng)(-.@)(-.dh)(-.z)(aa.aa)(-.l))
((1885 2605 14814)         present (\#.-)(-.z)(-.@)(p.d)(r.r)(e.e)(z.z)(-.dh)(@.-)[(n.n)(t.t)]))


((PATJLSB023 some parenchymal nuclei are present)
((   0  315  2248)            some (s.s)(@.o)(-.w)(m.m))
(( 315  935  7060)       bronchial (\#.-)[(b.b)(r.r)](o.u)(ng.ng)(k.k)(i.uu)(@.ng)(l.w))
(( 935 1600  8552)          nuclei (\#.-) [(n.n)(y.y)(uu.uu)(k.k)][(l.l)(i.i)](ai.ai))
((1600 1720 10446)              is (\#.-) (i.uu)(z.-))
((1720 2335 13707)         present (\#.-) (p.d)(r.r)(e.@)(z.z)(@.ng)[(n.n)(t.t)]))


((PATJLSB027 the atypical cells have abnormal nuclei)
((   0  790  4804)        atypical (-.v)(-.y)(-.y)(ei.ei)[(t.t)(i.i)][(p.d)(i.i)](k.k)[(@.-)(l.l))
(( 790 1340  6087)           cells (\#.-)(s.s)](-.@)(e.e)[(-.l)(l.l)(z.z)])
((1340 1495  7811)            have (\#.-)(h.h)[(@.@)(v.-))
((1495 2085 11108)        abnormal (\#.-)](a.a)(-.@)(b.-)(n.n)(-.l)(oo.oo)(m.m)[(@.-)(l.l))
((2085 2785 13325)          nuclei (\#.-)][(n.n)(y.y)(uu.uu)(k.k)](l.y)(i.@)[(ai.ai)(-.@)]))


((PATJLSB031 the cells have hyperchromatic nuclei)
((   0  210  1047)            this [(dh.dh)(i.i)(s.s))
(( 210  840  4675)          voided (\#.-)](v.-)(-.@)(-.l)(oi.oo)(-.z)(d.d)(@.@)(d.d))
(( 840 1725 10898)  hyperchromatic (\#.-)](h.-)(ai.ai)(p.d)(-.@)(-.ng)(@.@)(k.-)(r.r)(ou.uu)(m.-)
                                        (a.a)[(t.t)(i.i)](-.v)(k.-))
((1725 2385 14945)          nuclei [(\#.-)(n.n)(y.y)](uu.d)[(k.k)(l.y)](i.@)[(ai.ai)(-.@)](-.y)))

((PATJLSB035 the deposit contains small numbers of lymphocytes)
((   0  550  4580)         deposit (-.l)(d.n)(@.i)[(p.d)(o.o)](z.z)(i.i)(-.jh)(t.-))
(( 550 1145  5442)        contains [(\#.-)(k.k)(@.@)(n.n)(t.t)(ei.ei)(n.n)](z.n))
((1145 1590  6989)           small [(\#.-)(s.s)(-.dh)(m.m)](oo.oo)(l.-))
((1590 1870  9365)           large (\#.-)(l.-)(aa.aa)(jh.ng)[(-.@))
((1870 3105 15549)     lymphocytes (\#.-)](-.z)[(-.l)(-.l)(l.l)](-.@)[(i.-)(m.m)](f.f)[(ou.ou)
                                        (-.@)](s.s)(ai.ai)(t.k)(-.th)[s.s)(-.z)]))
```

```
((PATJLSB039 the flat sheets contain cohesive cells)
((   0  415  2558)            flat (-.l)(f.f)(l.l)(a.a)(-.@)(t.-))
(( 415  930  4628)          sheets [(\#.-)(sh.sh)](-.jh)(ii.ii)(t.t)[(s.s))
(( 930 1525  7505)         contain (\#.-)](k.g)[(@.i)(n.n)(t.t)(ei.ei)(n.n)](-.ng))
((1525 1840  9035)           clear (\#.-)[(k.k)(l.y)][(-.y)(i@.i)])
((1840 2815 13349)           cells [(\#.-)(s.s)(-.@)](-.v)(-.z)[(-.@)(e.o)][(-.l)(l.l)][(-.dh)
                                   (z.z)]))


((PATJLSB043 the preparation contains an undifferentiated carcinoma)
((   0   85   928)             the [(dh.dh)(@.@))
((  85  840  4321)     preparation (\#.-)](p.d)(r.r)(e.@)(p.d)(@.-)(r.r)(ei.ei)(sh.sh)(@.-)
                                   (n.n))
(( 840 1435  6373)        contains [(\#.-)(k.k)(@.@)(n.n)(t.t)](-.@)[(ei.ei)(n.n)(z.z))
((1435 1510  6830)              an (\#.-)][(@.@)(n.n)])
((1510 2580 13845) undifferentiated (\#.-)(uh.uh)(n.n)(d.d)(i.i)(f.f)(@.-)(r.r)(e.e)(n.n)(sh.sh)
                                   (i.i)(-.y)(ei.i)(t.t)(-.y)(@.-)(d.d))
((2580 3375 18423)       carcinoma [(\#.-)(k.k)](-.o)(-.@)(aa.-)[(s.s)(i.-)](n.n)(ou.ou)(m.m)(-.v)
                                   (@.@)))


((PATJLSB047 the smears contain collections of material)
((   0   95   847)             the [(dh.dh)(@.@))
((  95  625  1675)          smears (\#.-)](s.s)(m.m)[(i@.i)(-.@)](z.z))
(( 625 1110  3658)         contain (\#.-)](k.-)[(@.i)(n.n)(t.t)(ei.ei)](-.@)[(n.n))
((1110 2160 11563)         present (\#.-)](-.ng)(p.d)(r.@)(-.l)(e.e)(-.@)(-.jh)(-.ng)(-.ng)(z.z)
                                   (-.@)(-.ng)[(@.@)(n.n)(t.t)])
((2160 2545 14236)         pleural (\#.-)(p.-)(l.y)[(u@.uu)(r.r)](@.r)(-.@)(l.l)))


((PATJLSB051 the smears show malignant nuclei)
((   0   90   805)             the [(dh.dh)(@.@))
((  90  645  3038)          smears (\#.-)](s.s)[(m.m)(-.y)](-.@)[(i@.i)(-.@)](z.z))
(( 645  910  3935)          showed [(\#.-)(sh.sh)(ou.ou)](d.uu))
(( 910 1520  7087)       malignant (\#.-)(m.m)(@.@)[(l.l)(i.i)](g.ng)(n.n)[(@.@)(n.n)](t.-))
((1520 2195 11474)          nuclei [(\#.-)(n.n)(y.y)](uu.d)[(k.k)(l.y)](i.ou)(ai.ai)(-.y)))
((PATJLSB055 the specimen contains clumps of malignant cells)
((   0  115  1029)             the [(dh.dh)(@.@))
(( 115  755  1603)        specimen (\#.-)][(s.s)(p.d)(e.e)(s.s)(i.-)(m.m)](@.@)[(n.n))
(( 755 1350  3916)        contains (\#.-)](k.g)[(@.@)(n.n)(t.t)(ei.ei)](-.@)[(n.n)(z.z))
((1350 2485 10825)    bloodstained (\#.-)](b.d)(l.l)(uh.uh)(d.d)(s.s)(-.l)(t.g)[(-.l)(-.l)](-.dh)
                                   [(ei.ei)(n.n)](-.@)(d.d))
((2485 3145 12750)          cells [(\#.-)(s.s)(e.e)](-.@)[(-.l)(-.l)(l.l)](z.z)))
```

```
((PATJLSB059 the specimen is suboptimal)
((   0   90   781)              the [(dh.dh)(@.@))
((  90  700  1274)         specimen (\#.-)][(s.s)(p.d)(e.e)(s.s)(i.-)(m.m)](@.@)(n.n))
(( 700  770  2543)              are (\#.-)](-.@)(aa.-))
(( 770 1805  7756)       suboptimal [(\#.-)(s.s)](uh.uh)(b.-)(o.o)(p.k)(t.t)(-.@)[(i.-)(m.m)](-.l)
                                    [(@.-)(l.l)](-.dh)(-.v)))

((PATJLSB063 these bloodstained smears contain large numbers of inflammatory cells)
((   0  350  1867)            these (dh.v)(-.y)(ii.ii)(z.z))
(( 350 1055  5184)     bloodstained [(\#.-)(b.b)](l.-)(uh.aa)(d.d)(s.s)[(t.t)(ei.ei)](-.@)(n.n)
                                     (d.-))
((1055 1685  6261)           smears [(\#.-)(s.s)][(m.m)(-.y)(i@.i)(-.@)][(z.z))
((1685 2200  7714)          contain (\#.-)](k.d)[(@.@)(n.n)(t.t)(ei.ei)(n.n)])
((2200 2595  8568)            large [(\#.-)(l.l)(aa.aa)](jh.jh))
((2595 3405 15260)         dominant (\#.-)(d.n)(o.aa)(-.ng)(m.m)(-.@)(-.z)(i.i)(n.n)[(@.i)(n.n)]
                                    (t.k)(-.l))
((3405 3765 16646)            large [(\#.-)(l.l)(aa.aa)](-.@)(jh.jh))
((3765 4675 20984)         crystals (\#.-)(k.-)(r.r)[(i.i)(s.s)](-.@)(t.-)(@.@)[(-.l)(-.l)][(l.l)
                                    (-.z)(z.z)](-.dh)))

((PATJLSB067 these smears contain sheets of epithelial cells)
((   0  300  1635)            these (dh.dh)(-.y)(ii.ii)[(z.z))
(( 300  850  3051)           smears (\#.-)](s.s)[(m.m)(-.y)(i@.i)(-.@)][(z.z))
(( 850 1260  4713)          contain (\#.-)][(k.k)(@.i)(n.n)(t.t)(ei.ei)](n.-))
((1260 3265 15886)      mesothelial (\#.-)(m.m)(-.jh)(e.e)(-.v)(s.s)(-.@)(ou.ou)(-.v)(-.y)(-.th)
                                    [(th.th)(ii.ii)](-.y)[(-.l)(l.l)](-.z)(i.ou)(@.-)
                                    [(-.l)(-.l)(l.l)](-.z)))

((PATJLSB071 this pleural fluid contains moderate numbers of mesothelial cells)
((   0  220  1336)              this [(dh.dh)(i.i)(s.s))
(( 220  655  4495)          colloid (\#.-)](k.k)(o.o)(l.r)(oi.oo)(d.-))
(( 655 1200  7164)            fluid [(\#.-)(f.f)](l.l)(uu.uu)(-.@)(i.i)(d.d))
((1200 1785  9977)         contains [(\#.-)(k.k)(@.@)(n.n)(t.t)](-.i)[(ei.ei)(n.n)(z.z))
((1785 2470 14196)         moderate (\#.-)](m.m)(o.o)(d.d)(@.-)(r.r)(-.ng)(-.dh)(-.l)(@.o)(t.-))
((2470 3680 21275)      mesothelial (\#.-)(m.m)(-.dh)(-.z)(-.@)(-.v)[(e.e)(s.s)](ou.e)(-.dh)
                                    [(th.th)(ii.ii)][(l.y)(i.i)](-.w)[(@.-)(l.l))
((3680 4355 22726)            cells (\#.-)(s.s)](-.@)[(e.o)(-.l)][(l.l)(-.z)(z.z)]))
((PATJLSB075 this specimen is acellular)
((   0  125   888)              the [(dh.dh)(@.@)])
(( 125  800  2614)         specimen [(\#.-)(s.s)(p.d)(e.e)(s.s)](-.@)[(i.-)(m.m)(@.i)(n.n))
(( 800  970  2347)               is (\#.-)](i.i)[(z.z))
(( 970 1120  3321)             bare (\#.-)](b.-)(e@.ei))
((1120 1835  7039)            cells [(\#.-)(s.s)(e.e)](l.l)(-.@)[(-.dh)(z.z)](-.@)(-.v)(-.dh)))
```

```
((PATJLSB079 this voided urine contains some debris)
((   0  235  1078)            this [(dh.dh)(i.i)(s.s))
(( 235  585  3952)             wet (\#.-)](-.dh)(-.v)(-.w)(w.w)(e.e)[(t.t))
(( 585  855  5662)            poor (\#.-)](-.@)(p.d)(-.y)(u@.uu))
(( 855 1270  8417)           urine (\#.-)(y.-)[(u@.uu)(r.r)](i.@)(-.ng)[(n.n))
((1270 1820 11847)         contain (\#.-)](k.d)(@.@)(-.ng)[(n.n)(t.t)(ei.ei)(n.n))
((1820 2130 13459)            some (\#.-)](s.s)(@.o)(m.m)(-.ng))
((2130 2575 14943)          debris [(\#.-)(d.d)](e.e)(-.@)[(b.b)(r.r)](ii.ii)))
```

Bibliography

[Applebaum & Hanson 89] Ted H. Applebaum and Brian A. Hanson. Enhancing the discrimination of speaker independent hidden markov models with corrective training. In *International Conference on Acoustics Speech and Signal Processing.*, 1989.

[Baker 75] James K. Baker. The dragon system - an overview. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-23, 1975.

[Bellman 57] R. Bellman. *Dynamic Programming.* Princeton Univ. Press, 1957.

[Browning 91] S.R. Browning. Analysis of the phonmeme recognition performance of the arm continuous speech recognition system. *RSRE Memorandum No. 4463*, 1991.

[C Bourjot & Fohr 89] A. Boyer C Bourjot and D. Fohr. Phonetic decoding assessment. In *Eurospeech 89 conference proceedings*, 1989.

[Church 87] Kenneth W. Church. *Phonological Parsing in Speech Recognition.* Kluwer Academic Publishers, 1987.

[Cohen & Mercer 75] Paul S. Cohen and Robert L. Mercer. *Speech Recognition*, chapter The Phonological Component of a Automatic Speech-Recognition System. Academic Press, Inc., 1975.

[Gimson 70] A.C. Gimson. *An Introduction to the Pronunciation of English.* Edward Arnold (Publishers) Ltd., 1970.

[Godin & Lockwood 89] C. Godin and P. Lockwood. Dtw schemes for continuous speech recognition: a unified view. *Computer Speech and Language Journal*, 1989.

[Harrington *et al* 86] Jonathan Harrington, John Laver, and Doug Cutting. Word-structure reduction rules in automatic, continuous speech recognition. In *Proceedings of the Institute of Acoustics*, 1986.

[Harrington *et al* 87] Jonathan Harrington, Ian Johnson, and Maggie Cooper. The application of phoneme sequence constraints to word boundary identification in automatic, continuous speech recognition. In *Proceedings of the European Conference on Speech Technology, Edinburgh*, 1987.

[Jelinek 81] Frederick Jelinek. Self-organised continuous speech recognition. In *Automatic Speech and Recognition, Proceedings of the NATO Advanced Study Institute*, 1981.

[Klatt 77] D.H. Klatt. Review of the arpa speech understanding project. *Journal Acoustical Society of America*, Vol 62, no 6, 1977.

[Klovstad & Mondshein 75] J. W. Klovstad and L. F. Mondshein. The caspers linguistic analysis system. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-23, 1975.

[Knowles 87] Gerald Knowles. *Patterns of Spoken English.* Longman, 1987.

[L. R. Bahl & Mercer 88] P. V. de Souza L. R. Bahl, P. F. Brown and R. L. Mercer. A new algorithm for the estimation of hidden markov model parameters. In *International Conference on Acoustics Speech and Signal Processing.*, 1988.

[Lee 88] Kai-Fu Lee. *Large-Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System.* Unpublished PhD thesis, Carnegie Mellon University, 1988.

[Lowerre & Reddy 90] Bruce Lowerre and Raj Reddy. *Readings in Speech Recognition. Edited by Alex Waibel and Kai Fu Lee*, chapter The HARPY Speech Understanding System. Morgan Kaufmann Publishers Inc. San Mateo, California, 1990.

[Marino & Monte 90] Jose B. Marino and Enrique Monte. Generation of multiple hypotheses in connected phonetic-unit recognition by a modified one-stage dynamic programming algorithm. In *International Conference on Acoustics Speech and Signal Processing.*, 1990.

[McInnes *et al* 89] F.R. McInnes, Y. Ariki, and A.A. Wrench. Enhancement and optimisation of a speech recogni-

tion front end based on hidden markov models. In *Eurospeech 89 conference proceedings*, 1989.

[McKelvie 90] David McKelvie. Personal correspondence, 1990. CSTR, Univ. of Edinburgh.

[McKelvie 91] David McKelvie. From phonemes to sentences - lexical access in the integrated speech tecnology demonstrator. *CSTR Final Report*, 1991.

[M.J. Hunt & Mermelstein 80] M. Lennig M.J. Hunt and P. Mermelstein. Experiments in syllable based recognition of continuous speech. In *International Conference on Acoustics Speech and Signal Processing.*, 1980.

[Myers & Rabiner 81] C. S. Myers and L. R. Rabiner. A level building dynamic time warping algorithm for connected speech recognition. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-29, 1981.

[N. Sugumara & Furui 83] K. Shikano N. Sugumara and S. Furui. Isolated word recognition using phoneme-like templates. In *International Conference on Acoustics Speech and Signal Processing.*, 1983.

[Ney 84] Hermann Ney. The use of a one-stage dynamic programming algorithm for connected word recognition. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-32, 1984.

[Nowell 91] Peter Nowell. An efficient implementation of the n-best algorithm for lexical access. *2nd European Conference on Speech and Communication Technology. Genoa, Italy.*, 1991.

[Oshika *et al* 75] B.T. Oshika, R.V. Weeks V.W. Zue, H. Neu, and J. Aurbach. The role of phonological rules in speech understanding research. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-23, 1975.

[Picone *et al* 86] J. Picone, K.M. Goudie-Marshall, G.R. Doddington, and W. Fisher. Automatic text alignment for speech system evaluation. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-34, 1986.

[Ramsaran 90] Susan Ramsaran. *Studies in the pronunciation of English: A commemorative volume in honour of A.C. Gimson.* Routledge, 1990.

[Rohwer] Richard Rohwer. *Language Modelling for Automatic Speech Processing Ed. John Foster*, chapter N-gram. A Model for Statistical Processing of Data. Forthcoming.

[Rohwer 87] Richard Rohwer. A measure of deliberateness as an aid to the construction of grammars. In *Proc. of the European Conference on Speech Technology*, 1987.

[Sakoe & Chiba 78] H. Sakoe and S. Chiba. Dynamic programming algorithm optimisation for spoken word recognition. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-26, 1978.

[Sankoff & Kruskal 83] David Sankoff and Joseph B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and practice of Sequence Comparison*, chapter The Symmetric Time Warping Problem:

From Continuous to Discrete. Addison-Wesley, 1983.

[Schwartz & Chow 87] Richard Schwartz and Yen-Lu Chow. The n-best algorithm: An efficient and exact procedure for finding the n most likely sentence hypotheses. In *International Conference on Acoustics Speech and Signal Processing.*, 1987.

[Shockey & Reddy 73] L. Shockey and R. Reddy. Quantitative analysis of speech perception. In *Proc. Speech Communications Deminar, Stockholm*, 1973.

[Steinbiss 89] Volker Steinbiss. Sentence-hypotheses generation in a continuous-speech recognition system. In *Proc. of the European Conf. on Speech Communication and Technology*, 1989.

[Thompson 83] H.S. Thompson. Mchart: A flexible, modular chart parsing system. In *Proceedings of the Third Annual Meeting of the American Association for Artificial Intelligence, AAAI, Stanford, CA.*, 1983.

[Thompson 89] H.S. Thompson. A chart parsing realisation of dynamic programming with best-first enumeration of paths in a lattice. In *Proceedings of the European Conference on Speech Communications and Technology, European Speech Communications Association, Brussels*, 1989.

[Thompson 90] H.S. Thompson. Best-first enumeration of paths through a lattice: an active chart parsing solution. *Computer Speech and Language Journal*, 1990.

[Thompson *et al* 89] Henry S. Thompson, David McKelvie, and Fergus McInnes. Robust lexical access for continuous speech using dynamic time warping and finite-state transducers. In *Eurospeech 89 conference proceedings*, 1989.

[Vintsyuk 68] T.K. Vintsyuk. Speech discrimination by dynamic programming. *Kybernetika*, Vol. 4, 1968.

[Vintsyuk 71] T.K. Vintsyuk. Element-wise recognition of continuous speech composed of words from a specified dictionary. *Kybernetika*, Vol. 7, 1971.

[Waibel *et al* 89] A. Waibel, T. Habazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-37, 1989.

[Williams & Thompson 89] B. Williams and H. Thompson. Modelling phonological processes in continuous speech recognition. In *Eurospeech 89 conference proceedings*, 1989.

[Woods 76] W. Woods. Lexicon, lexical retrieval and control. *BBN Report No. 3438*, Vol. 3, 1976.

[Bridle 82] J.S. Bridle, M.D. Brown and R.M. Chamberlain An Algorithm for Connected Speech Recognition IEEE Trans. Acoustics, Speech and Signal Processing, May 1982